Detección de Armas en Vídeos Digitales
Trabajo
Fin de Grado

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
CURSO 2018–2019

PABLO ESTEVE CALZADO
ALEJANDRO MENDOZA SILVA

Directores

Luis Javier García Villalba
Ana Lucila Sandoval Orozco

Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Madrid, Junio de 2019

Agradecimientos

Pablo Esteve Calzado

En primera instancia me gustaría agradecer a mis tutores, Ana y Javier, la ayuda que me han dado durante todo el proceso de desarrollo del TFG.

Agradezco a mi compañero de TFG y gran amigo, Álex, quien ha hecho que el TFG sea divertido. Me llevo una gran amistad contigo.

Agradezco a mis amigos Pepe, Pelayo, Paimei, Noreña, Druet, Chemas, Neto, Martín, y muchos más, que han estado en los momentos difíciles y en los momentos felices. Han hecho en gran medida que sea la persona que soy y estaré eternamente agradecido.

Agradezco a mi familia el apoyo incondicional que me han demostrado a lo largo de toda mi vida. A mi otra ”madre”, por ser la mejor abuela del mundo. A Raya, por toda la felicidad que me diste. A mi hermano Alfonso, porque siempre me ha aportado persepectivas de la vida que nadie más me aporta. A mi hermano Jorge, porque consiguió que cambiara mi vida por completo. A mi padre, porque me enseña que hay muchas formas de querer, y ayudar no es siempre decir lo que se quiere oir. A mi madre, a quien le debo todo lo que he conseguido y lo que voy a conseguir en la vida, porque me ha demostrado que la felicidad se consigue con esfuerzo.

Quiero agradecer especialmente a Carolina todo lo que ha hecho por mi. Me haces ser feliz en todas sus vertientes. Todas las palabras son pocas para expresar los sentimientos que tengo hacia ti. Sin ti nada hubiera sido posible. Nada. Gracias por estar en los mejores y en los peores momentos. Te quiego.

Alejandro Mendoza Silva

Quiero dar las gracias en primer lugar a mis padres. Me sirvió de ayuda la confianza que ambos depositaron hacia mi. Mencionar a mi madre, que me convenció de que yo era capaz de hacer cualquier cosa que me propusiera. Esto fue un factor determinante y lo será a lo largo de mi vida tanto en mi desarrollo profesional como personal.

A mi padre por su inagotable energía que me proveyó de todo lo necesario para llevar a cabo mis estudios, con su ejemplo de esfuerzo y dedicación me motivo cada día a levantarme y poner empeño en el desarrollo de mi carrera. También agradecer a mí hermana por su continua preocupación por la progresión de mis estudios, sin ti tampoco hubiera sido posible esto.

Mencionar a mi compañero y amigo Pablo, sin ti esta aventura no habría sido la misma, espero que no sea la última y nuestros caminos vuelvan a juntarse.

Agradecer a mis tutores Ana y Javier por aceptar el reto y unirse con Pablo y conmigo a esta rama tan compleja y novedosa de la informática.

Quiero concluir diciendo que tanto mi padre, mi madre y mi hermana han sido, y serán, mis pilares en esta vida y me esforzaré para hacerles sentir orgullosos, para que cada momento que invirtieron en mi no sea en vano. Os quiero familia.

Índice General

Índice de Figuras

Índice de Tablas

Lista de Acrónimos

AA *Aprendizaje Automático*

AP *Aprendizaje Profundo*

DL *Deep Learning*

FP *Falso positivo*

FPS *Frames Per Second*

IA *Inteligencia Artificial*

ROI *Region Of Interest*

Abstract

This work has been done with the purpose of detecting handguns on videos using artificial intelligence algorithms, or more specific, deep learning. We have focused on handgun detection on videos, even though the techniques used are relevant to any other type of object like drugs, plates, faces, people...

There was a previous wide research to observe the techniques and models that are part of the state-of-the-art, comparing their results to know which ones are the better for the needs of this work.

For obtaining the best results, there are 5 experiments, where there were improvements on each part of the program. The first experiment was useful to choose the best model for the program, which was the Faster RCNN with the backbone Inception. The second experiment was an incremental improvement of the dataset, being four the number of improvements done in this experiment. The third experiment were changes on the model configuration, obtaining this way the parameters that better fitted to the handgun detection. In this experiment where was also an extension of the dataset.

For achieving this result it has been mandatory the initial research that was done over the object detection on the state-of-the-art. There was an improvement in the configuration of the model adapting it to the handgun detection. This has been possible thanks to the understanding of the multiple parameters that are part of the model.

One of the biggest accomplishments of this work, that might go unnoticed, is the dataset that was generated thanks to the manual labeling done within a program. This dataset consists of thousends of images of great relevance for handgun detection either on videos or images.

It was achieved a program which detects handguns on videos with a precision of 90 %, a recall of 92 % and an accuracy of 91 %.

Keywords: Artificial intelligence, machine learning, deep learning, computer vision, detection, videos, Faster RCNN, SSD, YOLO, Inception, Dataset.

Resumen

Este trabajo ha sido realizado con el propósito de detectar pistolas en vídeos mediante el uso de algoritmos de inteligencia artificial, en concreto, aprendizaje profundo. Está centrado en la detección de pistolas en vídeos, aunque las técnicas usadas se pueden aplicar a cualquier tipo de objetos como drogas, matrículas, caras, personas...

Se ha hecho una amplia investigación previa para observar las técnicas y modelos que forman parte del estado del arte, comparando los resultados de los mismos para saber cuáles son los mejores para las necesidades del trabajo.

Para la obtención de los mejores resultados, se ha decidido hacer 3 experimentos, en los que se han ido mejorando partes del programa. El primer experimento se centra en elegir el modelo que se va a usar, que finalmente es *Faster RCNN* con la red neuronal base *Inception*. El segundo experimento es una mejora incremental en la colección de datos, siendo 4 las mejoras. El tercer experimento son cambios en la configuración del modelo, obteniendo así los parámetros que más se ajustan a la detección de pistolas. En este último experimento se hace también una ampliación sobre la colección de datos, así como un balanceo en las imágenes.

Para conseguir este resultado ha sido imprescindible la investigación previa realizada sobre la detección de objetos en el estado del arte. Se ha conseguido mejorar la configuración de un modelo para adaptarlo a la detección de pistolas. Esto ha sido posible gracias al entendimiento de los múltiples parámetros que conforman un modelo.

Una de las grandes aportaciones de este trabajo, que puede pasar desapercibida, es la colección de imágenes que se ha generado gracias al etiquetado manual mediante un programa. Esta colección consta de miles de imágenes de gran relevancia para la detección de pistolas ya sea en vídeos o imágenes.

Se ha conseguido un programa que detecta pistolas en vídeos con una precisión del 90 %, una efectividad del 92 % y una exactitud del 91 %.

Palabras clave: Inteligencia artificial, aprendizaje automático, aprendizaje profundo, visión computacional, detección, vídeos, Faster RCNN, SSD, YOLO, Inception, colección de imágenes.

Capítulo 1

Introducción

Hoy en día existe mucha información sin explotar. En este caso, nos referimos a la información contenida en vídeos. Se genera tal cantidad de contenido digital, que es imposible supervisarlo con medios humanos. Es por esto que surge la necesidad del uso de técnicas de inteligencia artificial para procesar la información de forma masiva.

En este trabajo se va a aplicar el AP para la detección de pistolas en vídeos. Se probarán varios modelos, y se aplicarán mejoras sobre el modelo elegido para aumentar la detección. Para poder hacer esto se creará una colección de imágenes con etiquetas indicando dónde se encuentran las pistolas dentro de la imagen. Se hará esto de forma manual debido a la escasez de colecciones de imágenes de pistolas en internet.

1.1. Motivación

La detección de armas en vídeos es una problemática que aumenta diariamente, ya que afecta a la seguridad de las personas. Si bien es un problema importante, no se encuentran muchos programas que se dediquen exclusivamente a ello. De las aplicaciones que se han encontrado las colecciones de imágenes que usan son escasas y poco relevantes para el objetivo de este trabajo. Existen también trabajos relacionados con la detección de cuchillos en vídeos, pero no son aplicables a la detección de pistolas.

Desde luego queda mucho por avanzar en el campo de la detección de armas en vídeos, puesto que con los recientes modelos de detección de objetos, se puede hacer un software de detección de armas que sea más preciso que los anteriores.

1.2. Contexto

El presente Trabajo Fin de Grado se enmarca dentro de un proyecto de investigación titulado RAMSES aprobado por la Comisión Europea dentro del Programa Marco de Investigación e Innovación Horizonte 2020 (Convocatoria H2020-FCT-2015, Acción de Innovación, Número de Propuesta: 700326) y en el que participa el Grupo GASS del Departamento de Ingeniería del Software e Inteligencia Artificial de la Facultad de Informática de la Universidad Complutense de Madrid (Grupo de Análisis, Seguridad y Sistemas, http://gass.ucm.es, grupo 910623 del catálogo de grupos de investigación reconocidos por la UCM).

Además de la Universidad Complutense de Madrid participan las siguientes entidades:

- Treelogic Telemática y Lógica Racional para la Empresa Europea SL (España)
- Ministério da Justiça (Portugal)
- University of Kent (Reino Unido)
- Centro Ricerche e Studi su Sicurezza e Criminalità (Italia)
- Fachhochschule fur Offentliche Verwaltung und Rechtspflege in Bayern (Alemania)
- Trilateral Research & Consulting LLP (Reino Unido)
- Politecnico di Milano (Italia)
- Service Public Federal Interieur (Bélgica)
- Universität des Saarlandes (Alemania)
- Dirección General de Policía - Ministerio del Interior (España)

1.3. Objetivos y enfoque

El principal objetivo de este trabajo es la creación de un programa que sea capaz de detectar armas dentro de un vídeo, encuadrando el arma a lo largo del vídeo. El trabajo se centra en la detección de armas de fuego en vídeos mediante el uso de técnicas de inteligencia artificial (en nuestro caso usaremos el algoritmo *Faster RCNN* con la red neuronal base *Inception*), lo que implica su clasificación y localización dentro de la imagen.

Esto tiene muchas aplicaciones como la prevención de actos delictivos cuando se aplica el algoritmo a vídeos de videovigilancia. De esta forma se podría detectar un arma en una cámara de un aeropuerto y alertar a las autoridades de dónde se halla el sospechoso, qué tipo de arma porta, quién es, etc. También puede tener otras aplicaciones, como la revisión de vídeos para ver si contienen armas en el mismo, y en qué instante (al segundo) del vídeo se localiza el arma.

Se considera mas importante la precisión sobre la rapidez, ya que no puede permitirse que el software no detecte una arma, o que anuncie falsos positivos constantemente.

1.4. Plan de trabajo

El trabajo se divide en 4 fases: Investigación, implementación, experimentación y documentación. Se presentan las actividades contenidas en cada fase en la siguiente Tabla 1.1:

Tabla 1.1: Plan de trabajo

Tarea	**Duración**	**Fecha**$_{Inicio}$	**Fecha**$_{Fin}$
Investigación	87	01/10/2018	27/12/2018
Estudio de Python	20	01/10/2018	21/10/2018
Estudio de Aprendizaje automático	15	21/10/2018	05/11/2018
Estudio de Aprendizaje profundo	15	05/11/2018	20/11/2018
Estudio de modelos para detección de objetos	15	20/11/2018	05/12/2018
Lectura de trabajos de investigación	32	25/11/2018	27/12/2018
Implementación	120	20/12/2018	19/04/2019
Clasificador inicial	7	20/12/2018	27/12/2018
Colección de imágenes 1	7	27/12/2018	03/01/2019
Colección de imágenes 2	7	03/01/2019	10/01/2019
Colección de imágenes 3	10	10/01/2019	20/01/2019
Colección de imágenes 4	14	20/01/2019	03/02/2019
Configuraciones 1	25	03/02/2019	28/02/2019
Configuraciones 2	25	28/02/2019	25/03/2019
Colección de imágenes 5	25	25/03/2019	19/04/2019
Experimentación	128	27/12/2018	04/05/2019
Experimento 1	10	27/12/2018	06/01/2019
Experimento 2	38	06/01/2019	13/02/2019
Experimento 3	15	28/02/2019	04/05/2019
Documentación	180	20/11/2018	19/05/2019
Resúmenes	30	20/11/2018	20/12/2018
Recopilación información	120	20/12/2018	19/04/2019
Análisis experimentos	124	06/01/2019	10/05/2019
Redacción memoria	30	19/04/2019	19/05/2019

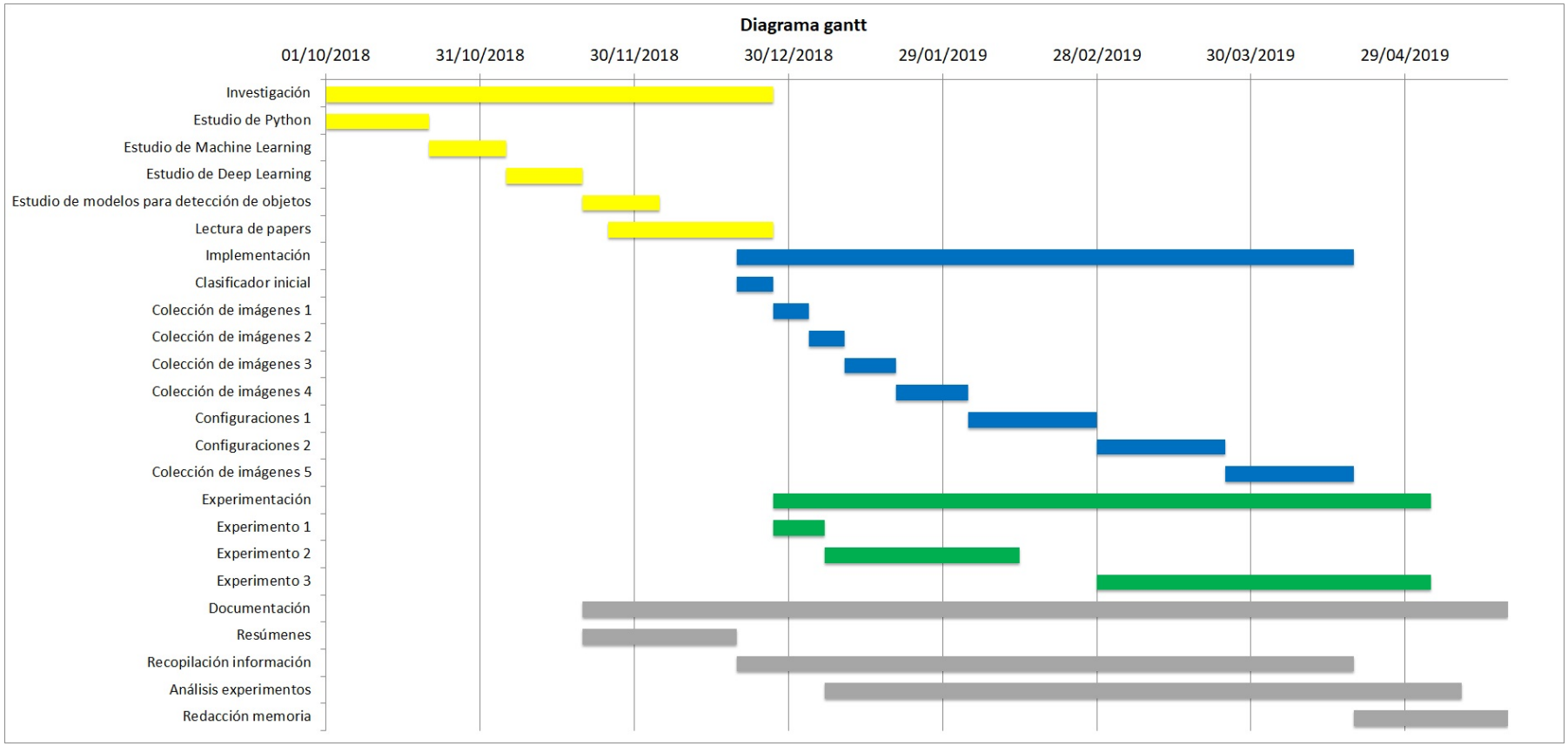


Figura 1.1: Diagrama de Gantt

1.5. Estructura de la Memoria

La memoria del proyecto está dividida en capítulos según la fase a la que corresponden, como a continuación se aclara:

En el Capítulo 1 se hace una breve introducción al proyecto, así como la moticación que lo impulsa.

En el Capítulo 2 se redacta un marco teórico de la inteligencia artificial para contextualizar el proyecto.

En el Capítulo 3 se muestra el Modelo usado en nuestro programa.

El Capítulo 4 se compone de varios experimentos en los que se ponen a prueba los desarrollos que se van haciendo, y se analizan los resultados para saber en qué línea continuar el desarrollo. También se muestran los resultados que se han obtenido en el trabajo.

El Capítulo 5 es la conclusión del trabajo y también se comentan las posibles mejoras que podría tener.

El Capítulo 6 es la introducción en inglés.

El Capítulo 5 es la conclusión y trabajo futuro en inglés.

El Capítulo 8 es el último capítulo, y describe cúal es el camino que ha llevado cada uno de los participantes en el proyecto. Se explican también las contribuciones que ha aportado cada uno.

Capítulo 2

Inteligencia Artificial

El ser humano siempre ha tratado de buscar nuevas formas de mejorar su condición de vida, desarrollando inventos que le faciliten tareas cotidianas. En cuanto a la IA, el objetivo ha sido construir máquinas que desarrollen procesos con inteligencia. Estos esfuerzos en un principio fueron destinados a la creación de autómatas. Para que reproduzcan acciones habituales de los seres humanos se podría enmarcar dentro de la IA. Estas máquinas están destinadas a facilitarnos la vida, no a competir con los seres humanos. Es el principio fundamental de la IA.

En general las máquinas permiten resolver problemas con la ayuda de algoritmos. Pero, ¿qué ocurre cuando se neesita resolver un problema que no se puede solucionar mediante un tratamiento algorítmico, como por ejemplo la clasificación de objetos por características comunes? En este caso se tiene la necesidad de dar otro punto de vista a la resolución de problemas, será necesario crear máquinas más versátiles. Y así es como surge el estudio de las capacidades humanas para generar diseños de nuevas máquinas que estarán enfocadas a reproducir esas capacidades.

El ser humano es capaz de realizar muchas operaciones simultáneas ya que el cerebro corresponde al de un sistema no-lineal, paralelo y complejo. El cerebro es un procesador eficiente, más que un computador, ya que tareas cotidianas para el cerebro resultan imposibles llevar a cabo mediante computación tradicional.

De este modo las redes neuronales emergen como modelo de diseño, se ha buscado emular el comportamiento del cerebro mediante estas redes. Estas redes son un procesador de información, con distribución paralela compuesto por unidades sencillas denominadas neuronas.

2.1. Historia de la Inteligencia Artificial

Se empezó a hablar de la IA en la primera mitad del siglo XX. Se hizo popular debido a las películas en las que aparecía un robot capaz de "pensar". Pero la idea que se tenía en ese momento de la IA no tenía nada que ver con la idea que se tiene hoy en día.

El precursor de la IA, al menos de forma teórica, fue Alan Turing, quien en un paper de 1950 [Tur50] explicaba cómo una máquina debería ser capaz de "pensar" como un humano.

En 1955-1956 Allen Newell, Cliff Shaw y Herbert Simon crearon el primer programa que hacía uso de la IA, *Logic Theorist* [Com19]. Este programa imitaba la habilidad de los humanos de resolver problemas. Tras la conferencia en la que presentaron dicho programa, aumentó significativamente la investigación en el campo de la IA y supuso un punto de inflexión decisivo.

A partir de este punto, y hasta 1970, la IA se hizo cada vez más importante. Los ordenadores tenían más capacidad de almacenamiento y eran más rápidos y baratos, por lo que esto favoreció el uso de ordenadores y a su vez de la IA. Llegó un punto en el que, tras superar los primeros obstáculos, se vio que todavía quedaba mucho por avanzar, y que se estaba muy lejos de conseguir lo que en un inicio se pensaba. Así como la capacidad computacional dejaba mucho que desear, lo mismo pasaba con las financiaciones. Fue decayendo poco a poco hasta 1980.

En 1980 la IA se potenció drásticamente debido a 2 factores importantes:

- Mayor financiación.
- Más y mejores algoritmos.

Hubo mucha financiación por parte de los gobiernos en la década de los ochenta, pero aún así no se lograron obtener los objetivos que se pretendían en el ámbito de la IA. Esto hizo que se frenara de nuevo la financiación, aunque hubo muchos científicos jóvenes con talento que fueron motivados en esa época.

A partir de 1990, sin financiaciones, se consiguió llegar a los anteriores objetivos planteados. En 1997 una IA [FH99] consiguió ganar al ajedrez al campeón actual del momento (Kasparov). En ese mismo año se creó un programa de reconocimiento del lenguaje natural [ZWL97]. Poco más adelante se creó un robot capaz de interpretar y expresar emociones [Bre01].

En 2006, Geoffrey Hinton empezó a usar el término AP[HOT06], y explica arquitecturas de redes neuronales con más capas que permiten un aprendizaje más profundo.

Hoy en día la IA se ve muy favorecida debido al creciente uso del "Big Data". La actual capacidad de computación de los ordenadores permite hacer muchos más cálculos, y seguirá aumentando en el futuro [Sch97].

2.2. Modelo Neuronal

La neurona artificial fue diseñada para reproducir las características de funcionamiento básico de la neurona biológica. Por lo que cada neurona tendrá una entrada y una salida. Modelar el comportamiento en conjunto de la red es el objetivo de un modelo nueronal. Para ello se obtienen las características más relevantes del comportamiento fisiológico de la neurona. En la Figura 2.1 se puede apreciar la estructura de una red neuronal.

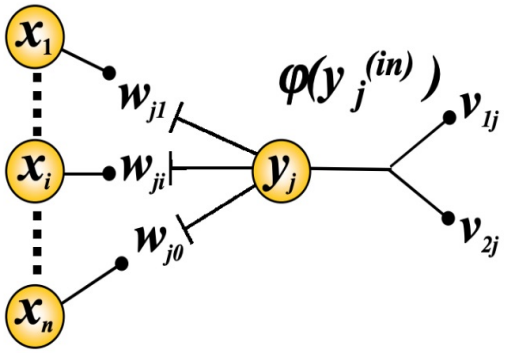


Figura 2.1: Red neuronal

Las neuronas X_n envían señales de entradas, los valores de W_n representan los pesos sinápticos. La función de estos pesos sinápticos es multiplicar su entrada correspondiente dando una importancia relativa a cada entrada. Todas las entradas ponderadas se suman y determinan el nivel de excitación de la neurona. Una representación vectorial del funcionamiento básico de una neurona artificial se indica mediante la siguiente expresión:

$$S = X * W$$

Siendo S la salida, X vector de entrada y W el vector de pesos.

La neurona se activa cuando la entrada total supera un cierto umbral. Se aplica una función de activación sobre y_j, que puede ser por ejemplo una función tipo sigmoide o tangente hiperbólica, como se observa en la Figura ??.

$$Salida = \frac{1}{1 + e^{-S}} \qquad \text{(Sigmoide)}$$

$$Salida = tanh(S) \qquad \text{(Tangente hiperbólica)}$$

El objetivo de las funciones es transmitir la idea de disparar sobre el umbral, ahora bien, muchas veces es deseable que la neurona se active con mayor dificultad. Por lo tanto eso implicaría subir el umbral, o hacer que la neurona se active con facilidad, que en este caso habría que bajar el umbral.

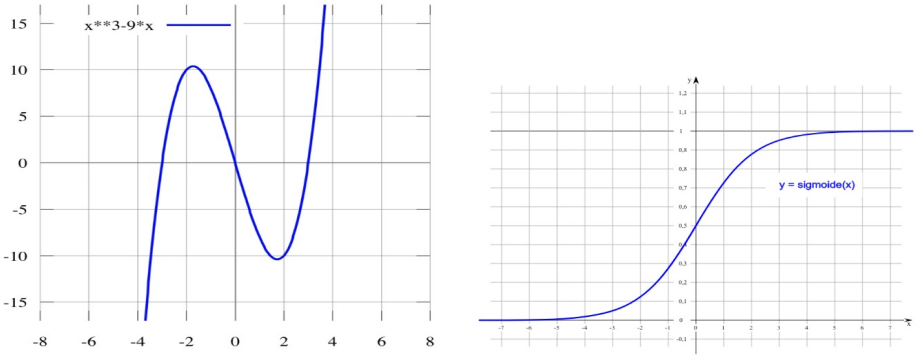


Figura 2.2: Tangente hiperbólica vs Sigmoide

2.3. Redes Neuronales Artificiales

La arquitectura de una red neuronal [RB01] se forma conectando múltiples procesadores elementales, siendo éste un sistema adaptativo que posee un algoritmo para ajustar sus pesos (parámetros libres) para alcanzar los requerimientos de desempeño del problema basado en muestras representativas. La capacidad de cálculo y potencia de la computación neuronal proviene de las múltiples conexiones de las neuronas artificiales que constituyen las redes neuronales artificiales.

Normalmente las redes más complejas y más grandes ofrecen mejores prestaciones en el cálculo computacional que las redes simples. El ordenamiento de las neuronas en capas o niveles imita a la estructura de capas que presenta el cerebro humano en algunas partes.

Es importante señalar que la propiedad más notable de las redes neuronales artificiales es su capacidad de aprender a partir de un conjunto de patrones de entrenamientos, es decir, es capaz de encontrar un modelo que ajuste los datos. El proceso de aprendizaje es también conocido como entrenamiento de la red.

2.4. Entrenamiento de las Redes Neuronales Artificiales

El aprendizaje [Mon19] es la parte que proporciona flexibilidad a una red neuronal y en esencia es el proceso por el que se adaptan las conexiones, para que la red responda de distinta forma a los estímulos. La red neuronal modifica sus pesos en función de una información de entrada. Los cambios que se producen durante el entrenamiento son la destrucción, modificación y creación de conexiones entre neuronas. En los sistemas biológicos naturales existe una continua destrucción y creación de conexiones entre las neuronas. Es curioso que las neuronas de la mayor parte de los seres vivos son esencialmente iguales, lo que diferencia a los humanos del resto de animales es la cantidad, organización y modo de cambio de las conexiones neuronales.

En los modelos de redes neuronales artificiales, la creación de una nueva conexión neuronal implica que el peso pasa a tener un valor distinto de cero. De esta manera, cuando una conexión se destruye su peso pasa a ser cero. Se da por finalizado el entrenamiento cuando los pesos de todas las conexiones de la red se mantienen estables y no tienen modificaciones.

Existen dos tipos de entrenamiento importantes:

2.4.1. Entrenamiento Supervisado

El entrenamiento supervisado consiste en pasar un vector de entrada a la red, calcular la salida de la red y compararla con la salida deseada. La diferencia entre ambas se utiliza para realimentar la red y modificar los pesos de acuerdo a un algoritmo que intenta minimizar el error.

El entrenamiento supervisado ha tenido mucho auge en varias aplicaciones. Sin embargo, ha tenido también muchas críticas ya que desde el punto de vista biológico no es muy lógico. No existe nada a nivel biológico en el cerebro que compare las salidas deseadas con las reales, por lo que no se ajusta a la realidad.

Existen tres maneras de llevar a cabo este tipo de entrenamiento:

2.4.1.1. Entrenamiento por Corrección de Error

Consiste en ajustar los pesos de las conexiones de la red en función de la diferencia entre los valores deseados y los obtenidos a la salida de la red, es decir, en función del error cometido en la salida.

Una aplicación de estos algoritmos lo constituye la regla del aprendizaje del Perceptrón. Cada neurona en la capa de salida calcula la desviación a la salida objetivo como error, luego se utiliza este error para cambiar los pesos sobre la conexión de la neurona precedente.

La regla del aprendizaje *Delta* o regla del mínimo error cuadrado, también utiliza la desviación a la salida objetivo, pero toma en consideración a todas las neuronas predecesoras que tiene la neurona de salida. Haciendo esto se puede cuantificar el error global cometido en cualquier momento del proceso del entrenamiento. Es necesario mencionar la regla del aprendizaje de propagación hacia atrás, la cual es una generalización de la regla de aprendizaje *Delta*. Esta regla permite realizar cambios en los pesos de las conexiones de la capa oculta.

2.4.1.2. Entrenamiento por Refuerzo

Se trata de un aprendizaje del tipo supervisado, que es más lento ya que no se dispone de un ejemplo completo del comportamiento que se busca. Es decir, durante el entrenamiento no se indica exactamente la salida que debería proporcionar la red.

2.4.1.3. Entrenamiento Estocástico

Consiste en realizar cambios aleatorios en los pesos de las conexiones de las neuronas y evaluar el efecto que tiene según el objetivo deseado. Se suele hacer una analogía con la energía: La energía es el grado de estabilidad de la red, de tal forma que el estado de mínima energía sería una situación en la que los pesos de las conexiones consiguen que su funcionamiento sea el más ajustado al deseado.

2.4.2. Entrenamiento no Supervisado

El conjunto de vectores de entrenamiento consta únicamente de vectores de entrada. El algoritmo de entrenamiento cambia los pesos de la red neuronal, de tal manera que produzca vectores de salida consistentes. El proceso de entrenamiento saca las propiedades comunes del conjunto de vectores de entrenamiento y agrupa en sectores los vectores parecidos.

2.4.2.1. Entrenamiento Hebbiano

Es un tipo de entrenamiento no supervisado. Pretende medir la correlación o extraer características de los datos introducidos en la entrada. El fundamento sobre el que se basa es: si dos neuronas N_i y N_j tienen el mismo estado simultáneamente (activo o inactivo), el peso de la conexión entre ambas neuronas aumenta.

2.4.2.2. Entrenamiento Competitivo y Comparativo

Otro tipo de entrenamiento no supervisado es el entrenamiento competitivo y comparativo, cuyo objetivo está orientado a la clasificación de los datos que se introducen en la entrada. Su principal característica es que si un patrón nuevo se determina que pertenece a una clase reconocida previamente, entonces este patrón hará que se modifique la forma de la clase.

2.5. Técnicas en la Inteligencia Artificial

Dentro de la IA existen varias técnicas que tienen distintos objetivos y metodologías. Estas técnicas están englobadas en la IA, por lo que comparten muchas características.

2.5.1. Aprendizaje Automático

El AA es la disciplina científica que permite a un programa aprender a tomar decisiones a partir de unos datos dados para así hacer predicciones. El AA hace uso de redes neuronales 2.3 para hacer las predicciones. Estas redes neuronales toman como entrada unos valores, pasan por un determinado número de capas, y luego dan como salida la predicción.

Se le dan ciertas reglas al programa, en forma de pesos en la red neuronal, para que interprete los datos de una manera determinada, pero el programa puede cambiar las reglas para aumentar la precisión en la predicción. Este método facilita las predicciones, ya que no es necesario saber la lógica que tiene que seguir el programa para que éste vaya mejorando. Un claro ejemplo es el filtro de spam de los correos electrónicos, ya que sin tener unas reglas completas sobre qué correos son spam y cuáles no, el algoritmo de predicción va mejorando con el tiempo.

Una de las características principales del AA es la capacidad de interpretar unas características dadas para sacar conclusiones. Estas características, que se proporcionan como entrada a la red neuronal, han de ser extraídas por un humano. Esta necesidad de proporcionar las características, hacen del AA una técnica con ciertas desventajas respecto al aprendiaje profundo.

Las ventajas del uso del AA respecto de otra técnica de IA es que, al usar redes neuronales, se le dota al programa de la capacidad de aprender con el tiempo. Una clara desventaja en cuanto al uso de esta técnica es la capacidad computacional que requiere respecto a otras técnicas más simples. Esta necesidad de capacidad computacional viene dada por el número de conexiones que se generan entre las neuronas de la red, y los algoritmos que han de ser ejecutados para el entrenamiento de los pesos de las conexiones en el entrenamiento.

2.5.2. Aprendizaje Profundo

El Aprendizaje Profundo está englobado dentro del AA, pero da un paso más. La extracción de características deja de ser competencia de un ser humano, y se introduce dentro de la arquitectura de la red neuronal. Esto hace que la complejidad de la arquitectura aumente, ya que es necesario el uso de un mayor número de capas.

El AP es una forma de dar libertad a un ordenador para decidir lo que es y lo que no es importante, ya que la extracción de características depende del mismo. En ocasiones esto es innecesario, pues las características a veces son fáciles de implementar. Aun así, en la mayoría de casos, el AP permite obtener resultados mejores que los humanos haciendo uso de su propia interpretación de lo que es importante y lo que no.

En relación a este trabajo, la dificultad de extraer de forma manual las características de una imagen es demasiado alta, por lo que el uso del AP es casi obligatorio. Gracias al AP no sólo se va a poder detectar el objeto, sino que se va a detectar sin tener que decirle a la red neuronal cuáles son las características del objeto de forma explícita.

La obtención de las características que hacen a una pistola ser una pistola las obtiene la red neuronal mediante el entrenamiento, que hace variar los pesos de las conexiones entre las neuronas de las capas que forman parte de la extracción de características.

La mayor ventaja del aprendizaje profundo respecto al AA es que no necesita supervisión humana para la extracción de las características. La desventaja del aprendizaje profundo, con mucha diferencia, es la inmensa capacidad computacional que puede llegar a requerir, dependiendo también de cuán sofisticada sea la red neuronal que se use.

2.6. Inteligencia Artificial vs Aprendizaje Automático vs Aprendizaje Profundo

Es importante destacar que el AP es una forma de llevar a cabo el AA, y que el AA está englobado dentro de la IA, tal y como se muestra en la Figura 2.3.

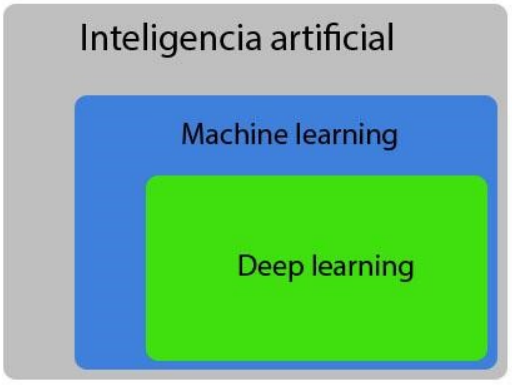


Figura 2.3: Inteligencia artificial vs aprendizaje automático vs aprendizaje profundo

Cuando se habla de diferencias entre AA e IA, se quiere hacer referencia a las características que tiene que cumplir el AA para serlo, y que no tiene por qué cumplir la IA. Por ejemplo, el uso de redes neuronales es obligatorio para el AA, aunque no lo es en la IA.

De esta forma se van a explicar las "diferencias" entre IA, AA y AP que se ven de forma resumida en la Tabla 2.1. Tanto la IA como el AA y el AP tienen como objetivo la toma de decisiones parecidas a un humano. En el AA y el AP es necesario el uso de redes neuronales, ya que son la base sobre la que se asientan estas técnicas. También en estas dos técnicas se tiene la capacidad de cambiar, mientras que en la IA no es algo intrínseco. La extracción de características automática es algo único del aprendizaje profundo, que aunque esté englobado en la IA y en el AA, sólo en el AP es necesario. Debido a esto, en el AP no se requiere supervisión humana, y esto lo "diferencia" de otras técnicas.

Debido a que la extracción de características en el AP se hace con una red neuronal, y a que se requieren más capas ocultas, la complejidad arquitectónica y la capacidad computacional requerida en el AP es necesariamente grande, mientras que en el AA no lo es necesariamente.

Tabla 2.1: AI vs AA vs AP

	AI	ML	DL
Toma decisiones como un humano	Sí	Sí	Sí
Uso de redes neuronales	No	Sí	Sí
Capacidad de cambiar	No	Sí	Sí
Extracción de características de forma automática	No	No	Sí
Supervisión humana	Sí	Sí	No
Complejidad de la arquitectura	Simple	Media	Compleja
Requiere alta capacidad computacional	No	No	Sí

En la Figura 2.4 se enseña de forma visual la diferencia que existe entre el AA y el AP en cuanto a la extracción de características.

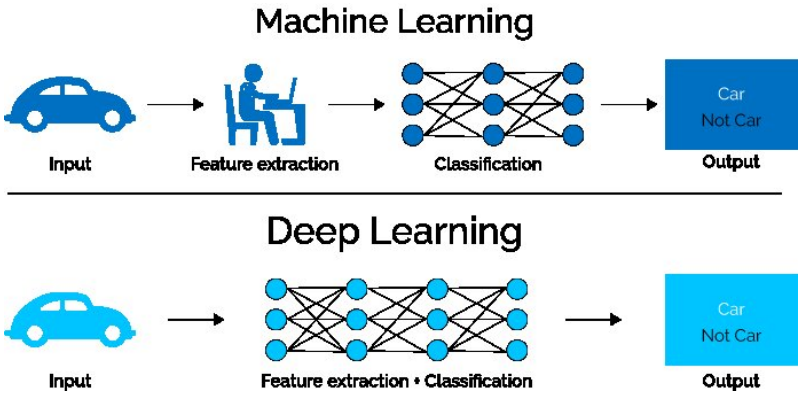


Figura 2.4: Aprendizaje automático vs aprendizaje profundo

2.7. Visión Artificial

Hay un concepto transversal al proyecto, que es la visión artificial. Este concepto se refiere a la capacidad de un ordenador de procesar una imagen dada, e interpretarla para sacar la características fundamentales y tener un entendimiento de lo que ocurre en la imagen. Usando técnicas de aprendizaje profundo, las características se sacan "automáticamente" con las capas profundas, por lo que el uso del aprendizaje profundo es lo más adecuado para este proyecto.

2.8. Clasificación de Objetos

La clasificación de objetos es un problema de visión artificial que consiste en interpretar una imagen y decidir a qué clase de las proporcionadas pertenece. Esta es una parte fundamental del proyecto, pues hay que decidir si en un *frame* de un vídeo hay un objeto o no, y de haberlo, cuál es.

Como se puede apreciar en la Figura 2.5, la imagen es la entrada de una red neuronal, que decide si es un pera o una manzana.

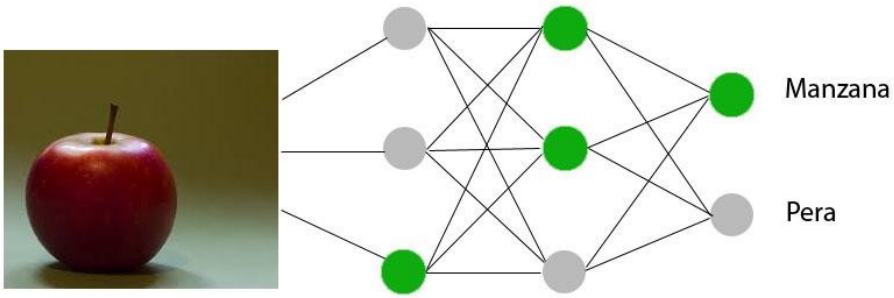


Figura 2.5: Clasificación de objetos

Para hacer clasificación en un conjunto de imágenes, se requiere entrenar el modelo con muchas imágenes etiquetadas con la clase a la que pertenecen. La clasificación puede ser decidir si una imagen pertenece a una clase o no, pero también puede ser decidir a qué clae, respecto de un conjunto de clases predefinidas, pertenece. La clasificación de imágenes con múltiples clases se puede abordar de varias formas, por ejemplo, en un trabajo [GN09] se hace una combinación de las características de las clases para la clasificación de las mismas.

2.9. Detección de Objetos

La detección de objetos va un paso más allá que la clasificación de objetos, y consiste en 2 partes: *Clasificación* y *Localización*. La detección de objetos tiene un coste mucho mayor que hacer sólo la clasificación, ya que no sólo se calcula a qué clase pertenece una imagen (1 parámetro) sino que también se calcula la posición en la que se encuentra el objeto dentro de la imagen (4 parámetros). Es por esto que la capacidad necesaria de procesamiento es mucho mayor que otro tipo de aproximaciones.

La detección de objetos tiene como objetivo no sólo detectar de forma correcta el objeto, sino evitar hacer falsas detecciones. Esto tiene mayor o menor importancia según el objetivo del programa que haga uso de esta técnica. En este trabajo se hace detección de pistolas en entornos de videos de seguridad y videovigilancia, por lo que es importante hacer la detección con el menor número de fallos posible.

Hay varios modelos que están diseñados para hacer la detección de la forma más eficaz y eficiente posible, aunque cada uno tiene sus características. Al ser la detección una problemática que consume muchos recursos, muchos modelos se centran en la rapidez en la detección, y no tanto en la precisión.

2.10. Técnicas de Detección de Objetos

La aplicación de las técnicas de detección de objetos más conocida es la detección de caras. En este contexto, se encuentra el detector de caras *PyramidBox* [TDHL18]. Este detector de caras tiene en cuenta el entorno para sacar conclusiones con mayor precisión, superando a otros detectores de cara del estado del arte en benchmarks reconocidos como FDDB [JLM10] y WIDER FACE [YLLT16]. Es mucho mejor que otros cuando la cara es pequeña o cuando la cara está borrosa en la imagen, ya que el contexto de la imagen juega un papel adicional, y consigue mejorar la precisión.

Como se observa en la Figura 2.6, la arquitectura del modelo es piramidal. Este tipo de implementación tiene implicaciones en los resultados de la red al analizar la imagen.

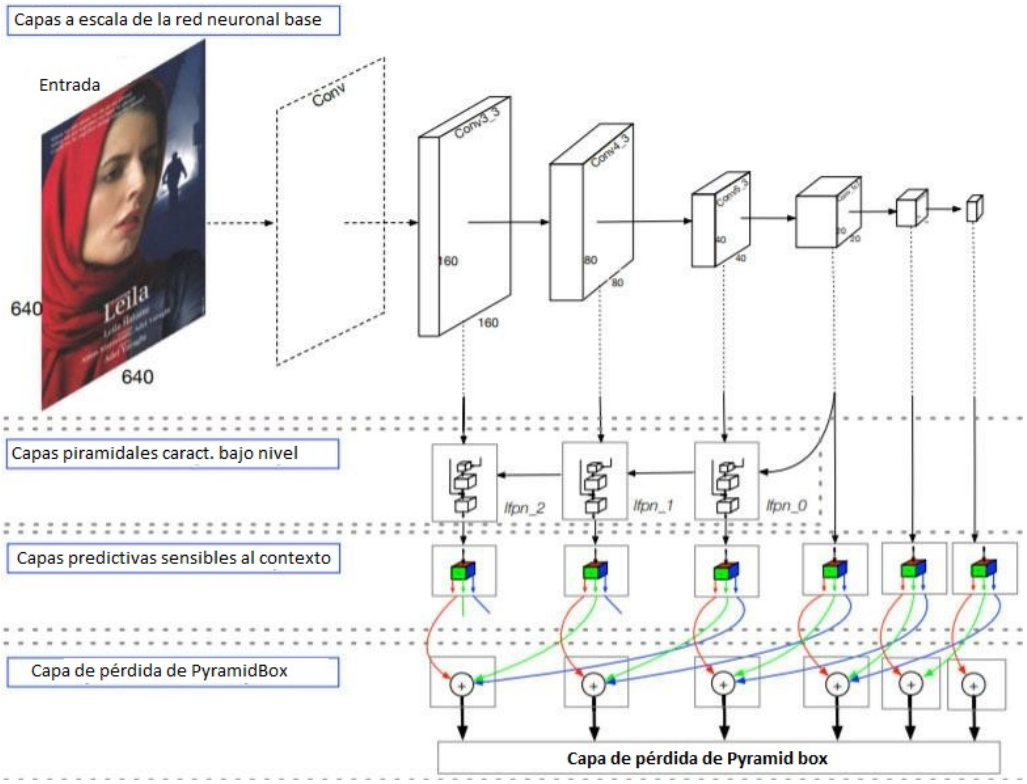


Figura 2.6: PyramidBox

Otro detector, aunque esta vez no es de caras, es el *RefineDet* [ZWB+18]. Este es un detector de una sola etapa que consiste en 2 módulos interconectados: el *Módulo de refinamiento de anclas* (ARM por sus siglas en inglés) y el *Módulo de detección de objetos* (ODM por sus siglas en inglés).

Para conectar un módulo con otro, se usan *Bloques de transferencia de conexión* (TCB por sus siglas en inglés), que convierten y transfieren la información del ARM al ODM. Esto se hace para reducir el espacio en el que se hace la clasificación y para afinar la localización y tamaño de las anclas o cajas.

Como se observa en la Tabla 2.2, los resultados que se obtienen con RefineDet son muy buenos. Consigue un punto intermedio (o incluso mejor) entre la rapidez de modelos como SSD o YOLO y la eficacia de modelos como Faster-RCNN. Otro modelo propuesto es el YOLOv3 [RF18], que es la última versión de YOLO. Tiene varias mejoras sobre el modelo original, y ahora tiene más capas, aunque eso no afecta a la rapidez.

Tabla 2.2: RefineDet

Modelo	**Red base**	**Tamaño entrada**	**# Cajas**	**FPS**	**mAP VOC 2007**	**mAP VOC 2012**
Two stage detectors						
Fast R-CNN	VGG-16	1000 x 600	2000	0,5	70 %	68,4 %
Faster R-CNN	VGG-16	1000 x 600	300	7	73,2 %	70,4 %
OHEM	VGG-16	1000 x 600	300	7	74,6 %	71,9 %
HyperNet	VGG-16	1000 x 600	100	0,88	76,3 %	71,4 %
Faster R-CNN	ResNet-101	1000 x 600	300	2,4	76,4 %	73,8 %
One stage detectors						
YOLO	GoogleNet	448 x 448	98	45	63,4 %	57,9 %
SSD300	VGG-16	300 x 300	8732	46	77,2 %	75,8 %
YOLOv2	Darknet-19	544 x 544	1445	40	78,6 %	73,5 %
SSD512	VGG-16	512 x 512	24564	19	79,8 %	78,5 %
RefineDet512	VGG-16	512 x 512	16320	24,1	81,8 %	80,1 %

La arquitectura del modelo se puede ver en la Tabla 2.3. Ya no usa la función *Softmax* para predecir la clase porque generaba problemas en la predicción en cajas en las que hay varias clases; ahora se usa una aproximación multiclase (*binary cross-entropy loss*). Incluye extracción de características usando 3 escalas distintas, que es parecido a las redes piramidales.

Algunos de los resultados más destacables son:

- Misma precisión media que modelos SSD, pero 3 veces más rápido (usando COCO).
- El tiempo de inferencia es una tercera parte del tiempo de inferencia de otras redes de detección del estado del arte.

Siguiendo con propuestas de modelo, una de las propuestas de modelo más innovadoras en el estado del arte es la de Spiking-YOLO [KPNY19]. Esta es una implementación de una Red Neuronal de Impulsos. En vez de usar las tradicionales redes neuronales profundas, se propone el uso de redes neuronales de impulso. Este tipo de redes tienen un carácter más realista, y su coste en tiempo es más elevado.

Normalmente estas redes se usaban sólo para la clasificación, pero en este caso se ha usado para la detección de objetos debido al aumento de la capacidad computacional que hay a medida que pasan los años. Es el primer detector de objetos que usa este tipo de red obteniendo resultados parecidos a otros detectores del estado del arte que usan redes profundas.

Tabla 2.3: YOLOv3

	Tipo	Filtros	Tamaño	Salida
	Convolucional	32	3 x 3	256 x 256
	Convolucional	64	3 x 3 / 2	128 x 128
	Convolucional	32	1 x 1	
1x	Convolucional	64	3 x 3	
	Residual			128 x 128
	Convolucional	128	3 x 3 / 2	64 x 64
	Convolucional	64	1 x 1	
2x	Convolucional	128	3 x 3	
	Residual			64 x 64
	Convolucional	256	3 x 3 / 2	32 x 32
	Convolucional	128	1 x 1	
8x	Convolucional	256	3 x 3	
	Residual			32 x 32
	Convolucional	512	3 x 3 / 2	16 x 16
	Convolucional	256	1 x 1	
8x	Convolucional	512	3 x 3	
	Residual			16 x 16
	Convolucional	1024	3 x 3 / 2	8 x 8
	Convolucional	512	1 x 1	
4x	Convolucional	1024	3 x 3	
	Residual			8 x 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

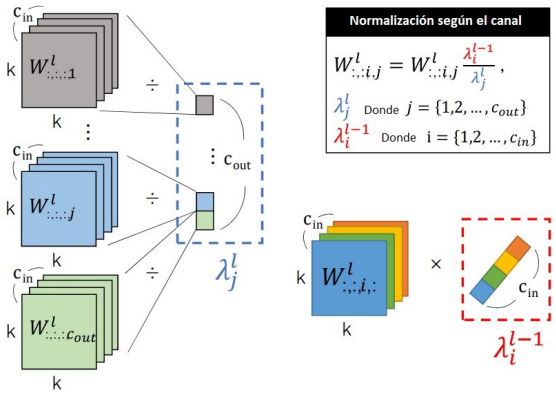


Figura 2.7: Spiking-YOLO

En la Figura 2.7 se ve la normalizacion según el canal que se propone; primero los pesos se normalizan con la máxima activación de cada canal para tener activaciones normalizadas, después se multiplica por λ^{l-1} las activaciones normalizadas para obtener de nuevo las activaciones originales.

Además de modelos, hay también propuestas de mejora de modelos, como es el caso de la *Detección con semántica enriquecida* [ZQX+18]. Es una mejora sobre los modelos de detección de objetos. Como se ve en la Figura 2.8, la arquitectura cambia y se le aplican mejoras que consisten en añadir una rama de segmentación semántica y un módulo de activación global, lo que hace mejorar los resultados en la detección.

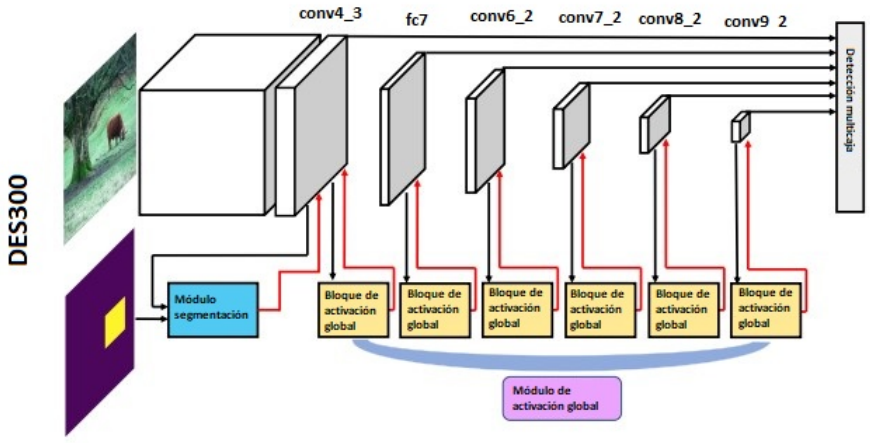


Figura 2.8: Detection with Enriched Semantics

En este caso usan el modelo SSD como base, y aplican el módulo de segmentación y el módulo de activación global para hacer mejoras semánticas a bajo nivel y mejoras en las capas de detección de alto nivel respectivamente.

Otro tipo de mejora sobre el SSD es la que se propone en [XZYA18], que surge de la necesidad de mejorar la precisión del modelo SSD en los objetos pequeños, ya que los detectores de una sola etapa tienen más problemas para detectar los objetos de menor tamaño. Este modelo demuestra que, usando la información que el contexto proporciona, se mejora sustancialmente la eficacia de la detección.

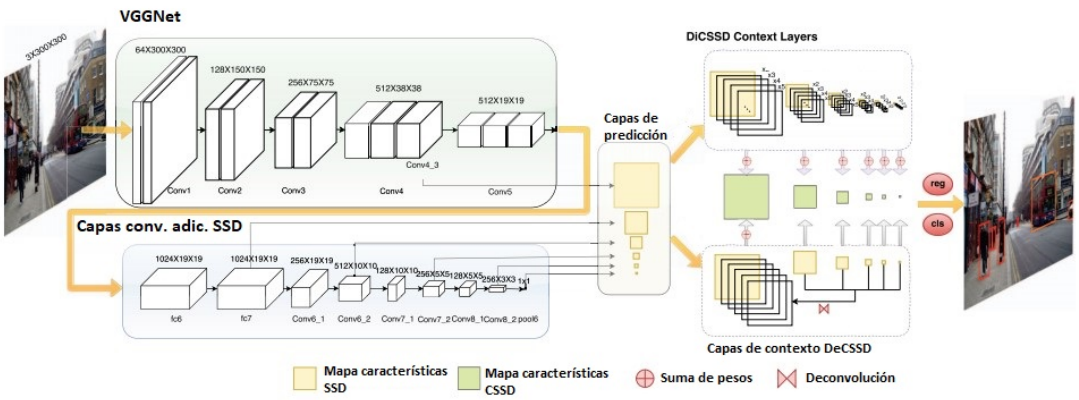


Figura 2.9: Context aware single shot detector

Como se ve en la Figura 2.9, esta mejora propone el uso de una red base *VGGNet* sobre la que se aplican capas convolucionales adicionales a las de SSD, que sirven como entrada para las capas de predicción. La salida de estas capas de predicción son la entrada de las capas de contexto, que son la mejora de la propuesta. Los resultados de aplicar este modelo dan un 3,2 % de mejora en la precisión media en los objetos pequeños comparado con el último modelo SSD, manteniendo la velocidad de procesamiento.

Otra de las posibles mejoras aplicadas a SSD es la que se propone en [KSL+19], que consiste en realizar una mejora consistente sobre el modelo SSD. Se enfoca en hacer coincidir las hipótesis de entrenamiento y la calidad de inferencia utilizando los anclajes refinados durante el entrenamiento.

Tabla 2.4: Consistent Optimization

Modelo	**Red base**	**AP**	**AP$_{50}$**	**AP$_{75}$**	**AP$_S$**	**AP$_M$**	**AP$_L$**
Two stage							
Faster R-CNN	ResNet-101	34.9	55.7	37.4	15.6	38.7	50.9
Mask R-CNN	ResNet-101	88.2	60.3	41.7	20.1	41.1	50.2
One stage							
SSD513	ResNet-101	31.2	50.4	33.3	10.2	34.5	49.8
YOLOv2	Darknet-19	21.6	44.0	19.2	5.0	22.4	35.5
YOLOv3	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9
RefineDet512	ResNet-101	36.4	57.5	39.5	16.6	39.9	51.4
ConRetinaNet	ResNet-101	40.1	59.6	43.5	23.4	44.2	53.3

Los resultados de aplicar la optimización en el modelo SSD con RetinaNet sobre COCO son visibles en la Tabla 2.4. Mejora la precisión media de 39.1 % a 40.1 %. Hay también propuestas de mejoras de otros modelos, como la que se propone en *Complex-YOLO* [SMAG18]. Es una red de detección de objetos basada en YOLOv2 que se centra en la detección de objetos en un entorno 3D (aplicable a vehículos, realidad aumentada, robots...).

Consiste en una Red de Propuestas de Región de Euler específica (E-RPN) para estimar la postura del objeto agregando una fracción imaginaria y real a la red de regresión.

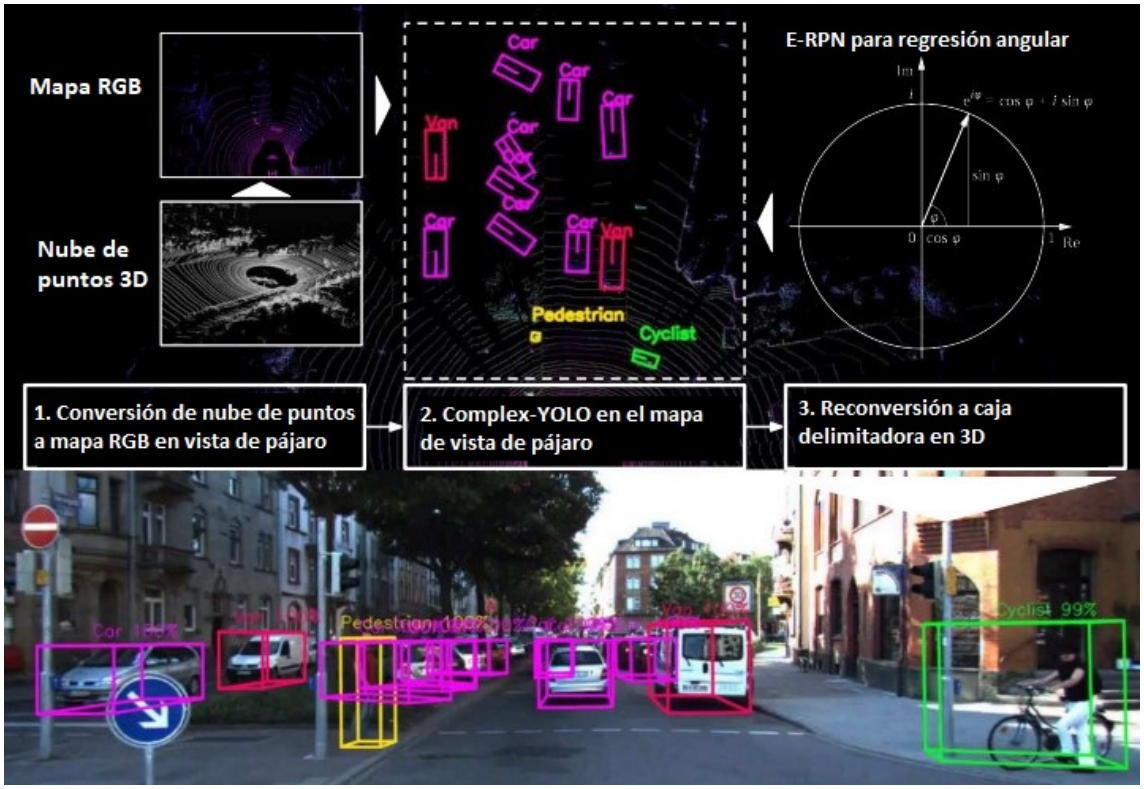


Figura 2.10: Complex-YOLO

Como se muestra en la Figura 2.10, la imagen inicial va pasando por una serie de cambios hasta hacer un cálculo de la estimación de la caja delimitadora en 3 dimensiones. Los resultados que se obtienen con este método superan significativamente a otros métodos del estado del arte en detección de objetos en 3D; especialmente en la velocidad de procesamiento.

Otra mejora para la detección en 3 dimensiones usando YOLO es Complexer-YOLO [SMA+19]. Es una red de detección de objetos en 3D que usa como base Complex-YOLO y realiza mejoras como añadir una puntuación de Escala-Rotación-Traslación (SRTs), que es una métrica de evaluación parametrizable que mejora la inferencia en un 20 % y reduce el tiempo de entrenamiento a la mitad. Usa también información temporal para mejorar la eficacia junto con un seguimiento de características en varios objetivos. Los resultados superan los FPS de otras redes de detección de objetos en 3D del estado del arte, pero disminuye mucho la eficacia, por lo que sería útil en caso de necesitar un detector en tiempo real.

Además de mejoras sobre modelos, y de propuestas de modelos, hay también propuestas de aplicación de los modelos muy interesantes, como puede ser la de [AmAaP+18]. Es un programa que detecta masas en mamografías. Usa la red YOLO y ha conseguido un 99,7 % de precisión en la detección de las masas. La clasificación que hace de dichas masas (benigna/maligna) tiene una precisión del 97 %. Los resultados de este programa se pueden ver en la Tabla 2.5.

Tabla 2.5: Detección mamografías

Modelo	# imágenes	Clases de predicción	Precisión detección	Precisión clasificación
LDA	DDSM (168)	Normal / Benigno / Maligno	86.00	78.57
QDA				76.19
NN				84.52
DBN				90.48
CNN	DDSM (2400)	Benigno / Maligno	X	96.70
YOLO	DDSM (2400)	Benigno / Maligno	99.7	97.00

Estos resultados no son extrapolables a otro tipo de detección de objetos, ya que la facilidad de detección de masas en mamografías es alta, pues no hay obstáculos en la visualización del objeto y el contexto es siempre igual.

Otra aplicación de la detección de objetos es un detector de armas en vídeos en tiempo real [OTH18], que es aplicado a cámaras de seguridad. Se hace uso del modelo Faster RCNN. Usan un dataset que han hecho público de 3000 imágenes en las que aparecen armas[PC19]. Un ejemplo de imagen de salida de este programa se aprecia en la Figura 2.11.

Figura 2.11: Alarma de detección de pistolas usando aprendizaje profundo

Hacen uso de 2 posibles aproximaciones para la detección de las armas:

- Ventana deslizante: Consiste en usar una ventana dentro de la imagen, e ir deslizándola, y considerar en cada paso la clasificación del objeto dentro de la ventana. Con esta aproximación se obtienen entre 10.000 y 100.000 posibles localizaciones para el objeto, por lo que su coste es altísimo.

- Propuestas de región: Esta aproximación tiene como objetivo la creación de un número determinado de propuestas de región sobre las que se aplicará la clasificación. Para obtener las propuestas se usa una red neuronal convolucional. En el caso de RCNN se crean unas 2000 propuestas de región. A pesar de ser un número que sigue siendo alto, reduce significativamente el tiempo con respecto a la aproximación anterior.

Para configurarlo como una alarma, hace que se active la alarma cuando hay 5 fotogramas seguidos en los que la red dice que hay un arma, para así evitar un gran número de falsas alarmas cuando se detecta un arma de forma equívoca.

Sobre este programa se hizo una mejora [CTP+19] que consiste en mejorar el preprocesamiento del dataset.

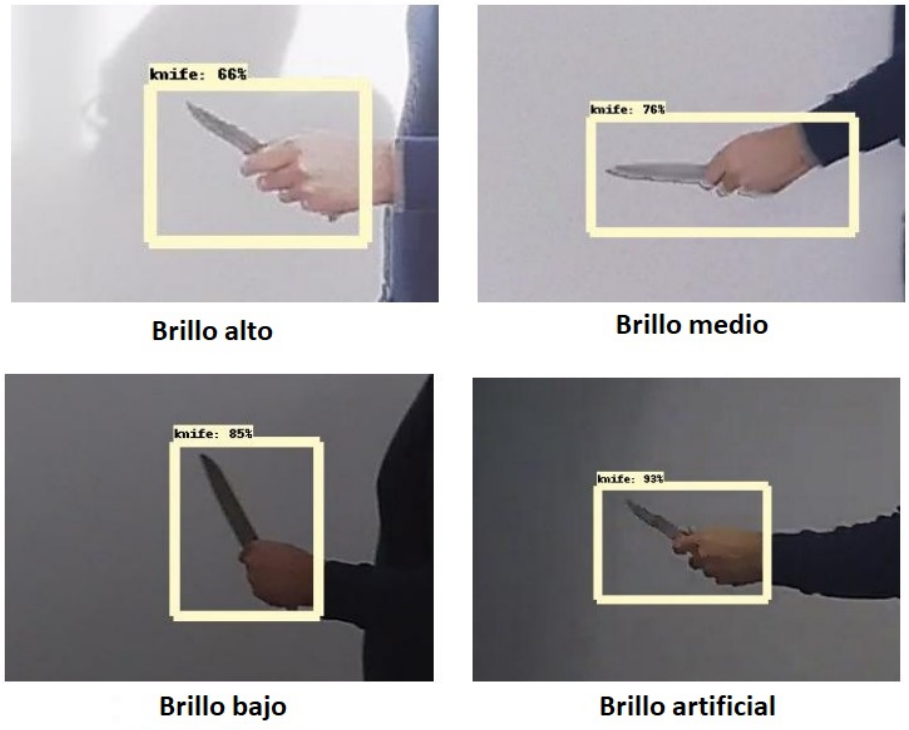


Figura 2.12: Preprocesamiento de vídeos para detección de armas

En este detector se centran en mejorar el preprocesado de los vídeos en los que se van a detectar objetos de metal, como pueden ser cuchillos o pistolas. Se hacen muchas mejoras cambiando el brillo y modificando el vídeo antes de procesarlo. Se llega a la conclusión de que cambiando el brillo de una imagen, cambia la probabilidad de detección del objeto dentro de la misma, como se puede apreciar en la Figura 2.12. Se comprueba que a mayor brillo, peor es la detección sobre elementos de metal, por lo que será positivo para la precisión disminuir el brillo en el preprocesado del dataset.

2.11. Modelos

El TFG se centra en hacer un programa que detecte armas en vídeos usando AP. Se pueden dividir los modelos basados en AP en dos tipos: Detectores de una etapa y detectores de dos etapas.

2.11.1. Detectores de una Etapa

Obtienen las RoI de la imagen mediante una búsqueda de las mismas. Esta búsqueda se hace con anterioridad a la clasificación de las RoI. Por lo que el modelo se divide en 2 partes diferenciadas, la fase de búsqueda de las RoI y la fase de clasificación de las RoI. La primera fase alimenta a la segunda.

2.11.1.1. Detector de Un Solo Vistazo

Este modelo usa redes convolucionales de principio a fin para hacer la detección. El modelo, como se ve en la Figura 2.13, toma un imagen como input, y la pasa por múltiples capas de una red neuronal del tipo "feed-forward".

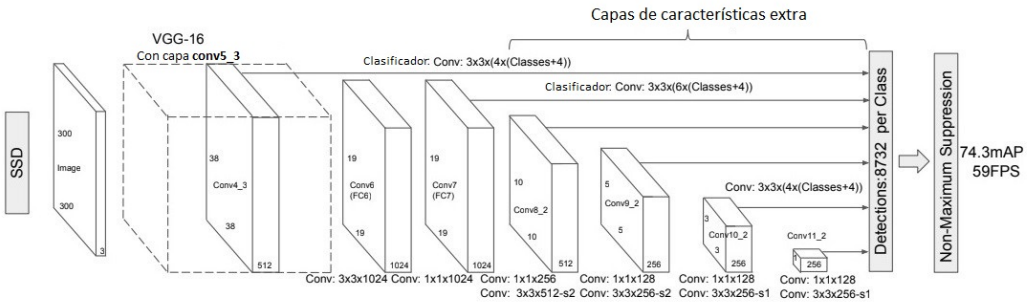


Figura 2.13: Detector de un Solo Vistazo

El *detector de un solo vistazo*, o SSD por sus siglas en inglés (Single shot detector), es un detector de una sola etapa. Las primeras capas de la red están basadas en una arquitectura estándar usada para la clasificación de imágenes de alta calidad. Después se añaden más capas para hacer la detección usando las siguientes características clave (key features):

- Mapas de características multi-escala para la detección: Se añaden capas de características convolucionales al final de la arquitectura de clasificación mencionada anteriormente. Las capas van disminuyendo su tamaño de forma progresiva para permitir la detección en múltiples escalas.
- Predictores convolucionales para la detección: Cada capa de características añadida produce un conjunto de predicciones de detecciones usando un conjunto de filtros convolucionales. Esto se hace antes de la arquitectura de red SSD.

- Cajas predeterminadas y ratios de aspecto: Se asocia un conjunto predeterminado de cajas delimitadoras con cada celda del mapa de características. Las cajas predeterminadas seccionan el mapa de características de forma convolucional, para que la posición de cada caja en relación a su celda sea fija. Para cada celda del mapa de características, se predicen los valores relativos a las formas de las cajas predeterminadas en la celda, así como las puntuaciones que indican la presencia de cada clase en cada una de las cajas. Es decir, que para cada 1 de las K cajas en una posición dada, se computan C puntuaciones de clases y los 4 valores relativos a la forma de la caja original. Esto resulta en un total de $(C+4)*K$ filtros aplicados en cada posición del mapa de características, resultando en $(C+4)*K*M*N$ salidas para un mapa de características de $M*N$.

2.11.1.2. Solo Miras Una Vez

El modelo Sólo miras una vez, o YOLO por sus siglas en inglés (You only look once) es otro detector de una sola etapa. Anteriores sistemas de detección se basaban en clasificadores o localizadores para hacer la detección. Aplicaban el modelo a la imagen en múltiples sitios y distintas escalas, y las regiones con una alta puntuación se consideraban detecciones.

YOLO funciona de una forma completamente distinta. Se aplica una sola vez la red neuronal a toda la imagen. Esta red divide la imagen en regiones y predice las cajas delimitadoras y las probabilidades para cada región. Estas cajas se ponderan con las probabilidades predichas.

Este modelo, que se aprecia en la Figura 2.14 tiene varias ventajas respecto a sistemas basados en la clasificación. Observa toda la imagen de una sola vez en el tiempo de prueba, por lo que sus predicciones se informan por el contexto general de la imagen. Hace también predicciones con una sola evaluación de la red neuronal a diferencia de sistemas como el RCNN, que requiere de miles de evaluaciones para una sola imagen. Esto lo hace extremadamente más rápido.

Al igual que el modelo SSD, éste es un detector de una etapa, por lo que es rápido, y su precisión es algo peor. Debido a la forma en que el modelo está hecho [RDGF16], le es mucho más difícil detectar objetos pequeños dentro de una imagen. Este inconveniente es muy grande en este proyecto ya que, al estar enfocado en la detección de armas (normalmente en cámaras de seguridad), va a tener gran dificultad para detectar con precisión las armas (que son pequeñas) en las imágenes.

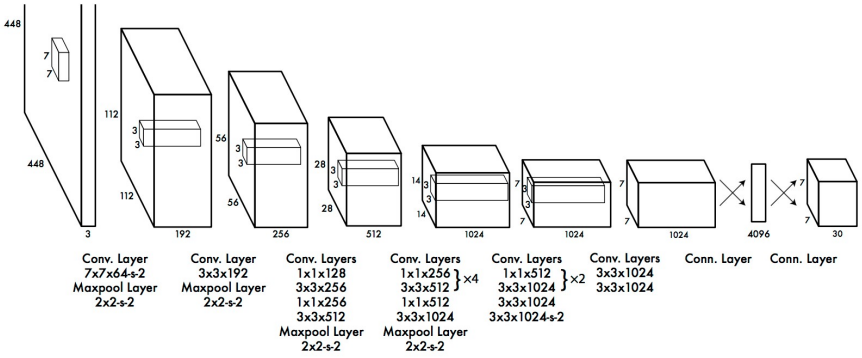


Figura 2.14: YOLO

- El modelo. El modelo del sistema de detección es como un problema de regresión. Divide la imagen en una cuadrícula de $S * S$ y para cada celda de la cuadrícula predice B cajas delimitadoras, la confianza para esas cajas, y C probabilidades para las clases. Estas predicciones se codifican como un tensor $S * S * (B * 5 + C)$. Esto se ve claramente en la Figura 2.15.

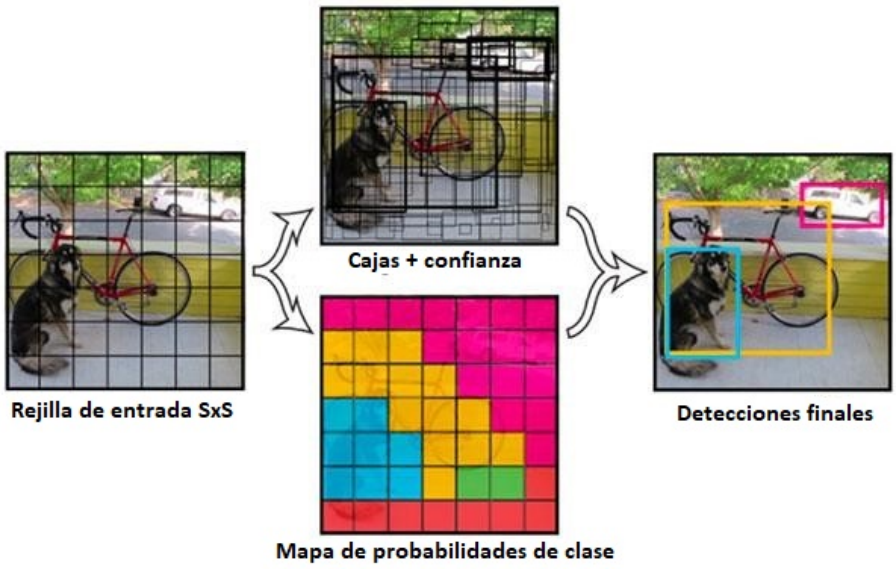


Figura 2.15: YOLO

- Limitaciones. YOLO impone fuertes restricciones espaciales en las cajas delimitadoras debido a que cada celda de la cuadrícula sólo predice 2 cajas y sólo puede tener una clase. Esta restricción espacial limita el número de objetos cercanos que el modelo puede predecir. El modelo tiene problemas con objetos pequeños que aparecen en grupo.

Como el modelo aprende a predecir cajas delimitadoras a partir de datos, le cuesta generalizar con objetos con un nuevo o inusual ratio de apariencia o configuración.

Finalmente, cuando se entrena con una función de pérdida que mejora el rendimiento de detección, la función de pérdida trata errores por igual en cajas pequeñas y cajas grandes. Un pequeño error en una caja grande, no es significativo, pero un error pequeño en una caja pequeña, tiene un efecto mayor.

La principal fuente de errores en YOLO son las localizaciones incorrectas.

2.11.2. Detectores de Dos Etapas

Los detectores de dos etapas se saltan la fase de búsqueda de RoI y alimenta la red directamente con la imagen.

2.11.2.1. Red Neuronal Convolucional Basada en Regiones

El modelo *Red neuronal convolucional basada en regiones*, o RCNN por sus siglas en ingles (Region-based convolutional neural network) es un detector de dos etapas. Para solventar el problema de seleccionar un gran número de regiones, Ross Girshik [GDDM14] propuso un método donde se usa una búsqueda selectiva para extraer 2000 regiones de una imagen, y las llamó propuestas de región. Usando este método, en vez de extraer muchísimas propuestas, trabajas con 2000 propuestas de región exactamente. El algoritmo de búsqueda selectiva es el siguiente:

- Se generan sub-segmentaciones iniciales (regiones candidatas).
- Usar el algoritmo *greedy* para combinar de forma recursiva regiones similares en regiones más grandes.
- Usar las regiones generadas para producir las propuestas de región finales.

Estas 2000 propuestas de región se juntan y se meten en una red neuronal convolucional que produce un vector de características de 4096 dimensiones como salida. La red neuronal convolucional actúa como un extractor de características.

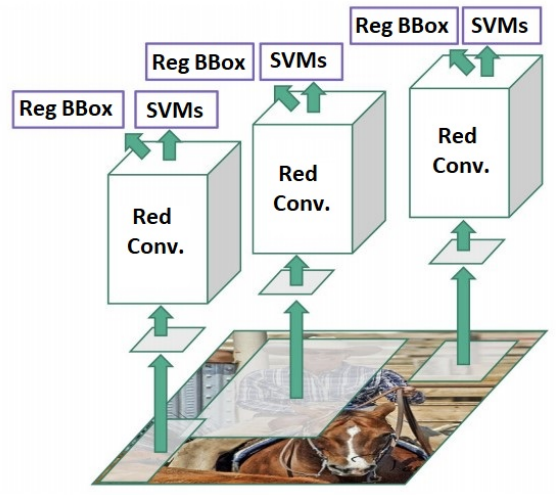


Figura 2.16: RCNN

Estas características se introducen como entrada a una *Máquina vectorial de soporte*, o SVM por sus siglas en inglés (support vector machine), para clasificar la presencia de objetos dentro de la propuesta de región, como se ve en la Figura 2.16. Además de predecir la presencia de un objeto en las propuestas de región, el modelo predice también 4 valores que dicen la posición y tamaño de la caja para aumentar la precisión de la caja delimitadora.

Problemas con RCNN:

- Consume mucho tiempo en entrenar la red porque clasifica 2000 propuestas de región por cada imagen.
- No puede ser implementado en tiempo real ya que tarda unos 47 segundos por cada imagen de test.
- El algoritmo de búsqueda selectiva es un algoritmo fijado. Por lo tanto, no hay aprendizaje en esa parte, y podría desencadenar malas propuestas de región.

2.11.2.2. Red Neuronal Convolucional Rápida Basada en Regiones

El modelo *Red neuronal convolucional rápida basada en regiones*, o Fast RCNN por sus siglas en inglés (Fast Region-based Convolutional Neural Network), usa como base el algoritmo RCNN mostrado en el apartado anterior, pero con diferencias y mejoras que se explican a continuación.

En vez de generar una pirámide de capas, Fast RCNN deforma las regiones de interés en una sola capa usando *RoI pooling*. El *RoI pooling* usa *Max pooling* para convertir las características contenidas en una región de interés en un mapa de características pequeño de dimensiones HxW.

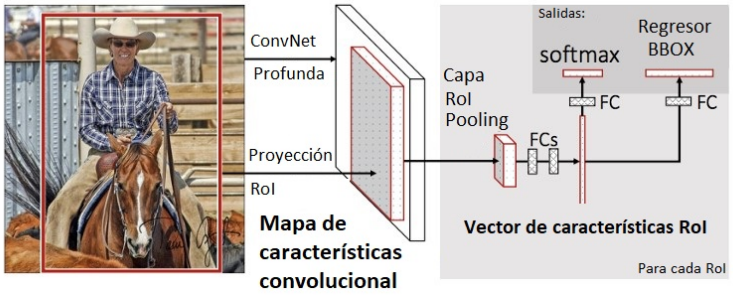


Figura 2.17: Fast RCNN

Se puede decir que Fast RCNN es un caso especial de *SPPNet*, pero en vez de usar múltiples capas, Fast RCNN usa solo una. Esto se puede ver claramente en la Figura 2.17. Esta capa alimenta a un red completamente conectada para la clasificación usando regresión lineal y *softmax*. La caja delimitadora es después refinada con regresión lineal.

En Fast RCNN, todos los parámetros dentro de la red neuronal convolucional pueden ser entrenados juntos. Estos parámetros se entrenan juntos con una *función logarítmica de pérdida* de la clasificación de clase y con una *función de pérdida L1* de la predicción de la caja delimitadora.

2.11.2.3. Red Neuronal Convolucional Más Rápida Basada en Regiones

El modelo *Red neuronal convolucional más rápida basada en regiones*, o Faster RCNN por sus siglas en inglés (Faster Region-based Convolutional Neural Network), está basado en el modelo Fast RCNN que se explica en el apartado anterior. A continuación se explican diferencias y mejoras respecto al mismo.

Tanto el modelo RCNN como el Fast RCNN usan la búsqueda selectiva para obtener las propuestas de región. Esta búsqueda es muy lenta y consume mucho tiempo, afectando al rendimiento de la red. Por esto, Shaoqing Ren propuso usar una red convolucional que sirviera para hacer las propuestas de región, ahorrando así mucho tiempo y mejorando la eficiencia del modelo, que se puede observar en la Figura 2.18.

La diferencia entre Fast RCNN y Faster RCNN es que no se usa un método de propuestas de región para crear las propuestas de región. En vez de eso, se entrena una red de proposición de regiones que coge los mapas de características como entrada, y da como salida las propuestas de región. Estas propuestas alimentan la capa *RoI pooling* en la Fast RCNN.

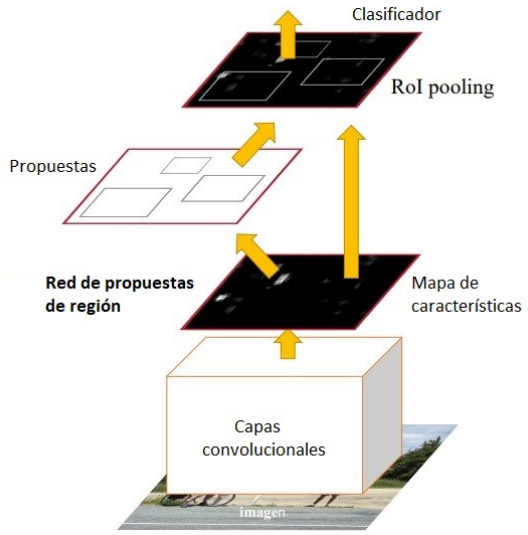


Figura 2.18: Faster RCNN

2.11.2.4. LSTM

Este modelo está englobado en las arquitecturas recurrentes, que se basan en poder acceder a información del pasado, y no solo a la del presente. Esto permite a este tipo de redes tomar mejores decisiones teniendo en cuenta las tomadas en el pasado.

La principal idea de la arquitectura de este modelo, como se ve en la Figura 2.19, es el uso de una celda de memoria que mantiene su estado a lo largo del tiempo, y unidades de acceso no lineales que regulan el flujo de información hacia dentro y hacia fuera de la celda de memoria. Los últimos estudios añaden nuevas mejoras a la arquitectura LSTM original [HS95] [HS97].

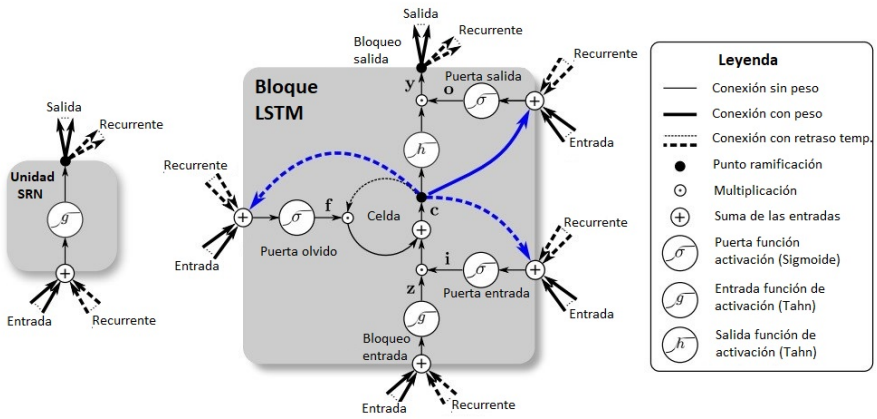


Figura 2.19: LSTM

La importancia del modelo LSTM es que es un modelo que usa una red neuronal recurrente, lo que implica que los hechos sucedidos en el pasado se tienen en cuenta en el futuro. Es por esto que son muy interesantes a la hora de tratar con vídeos o con problemáticas como la interpretación de voz a texto. Sin embargo, al ser de reciente creación, no está bien documentada, y todavía no está bien aplicada en vídeos. Además, la capacidad de procesamiento que requiere es demasiado alta para poder ver resultados con la capacidad de computación disponible para la realización de este trabajo.

Capítulo 3

Modelo de Detección de Armas en Vídeos Digitales

Este trabajo se compone de dos partes bien diferenciadas: La colección de imágenes y el modelo. A continuación se muestra cómo están formadas estas dos partes, explicando en profundidad las características de cada una de ellas.

3.1. Colección de Imágenes

Una colección de imágenes es un conjunto de imágenes con sus etiquetas asociadas, en este trabajo, las etiquetas indican la posición y la clase del objeto dentro de la imagen. La colección de imágenes se usa para entrenar y para testear el modelo.

En este trabajo la colección de imágenes se divide siempre en la proporción 80-20 para entrenamiento-pruebas. Se elige esta proporción ya que es la más recomendada para no malgastar muchas imágenes en las pruebas, pero que permita al programa sacar conclusiones de los resultados de la misma.

Las características que hacen de una colección de imágenes ser buena son las siguientes:

- Tener muchas imágenes.
- Tener muchos escenarios variados en los que se usan las armas.
- Que haya muchas posiciones distintas de las armas.
- Que haya más o menos la misma cantidad de imágenes por escenario.
- Que haya más o menos la misma cantidad de imágenes por posición.
- Que las imágenes vengan de vídeos parecidos a los que se van a usar para la detección.
- Que la cantidad de imágenes con arma, y sin arma sea balanceada.

3.1.1. Características

La colección de imágenes que se utiliza contiene imágenes que se han adquirido y que se han creado a lo largo del segundo experimento, y de una parte del tercer experimento.

La colección de imágenes consta de 1200 imágenes de pistolas seleccionadas de una colección de 3000 imágenes. 120 imágenes de pistolas siendo usadas en el contexto que se necesita. *Fotogramas* relevantes de 9 vídeos de atracos. *Fotogramas* relevantes de vídeos caseros hechos para mejorar la detección.

Forman también parte del dataset muchas imágenes sin pistola hasta llegar a la proporción 50-50 de *Pistola-No pistola*.

3.2. Configuración del Modelo

El modelo se compone de 3 redes neuronales. La primera red se encarga de la extracción de características. Esta red es la red neuronal base, que en este caso es la *Inception*. La segunda y tercera red pertenecen a Faster RCNN. Faster RCNN se divide en estas dos redes en las que la primera red tiene como objetivo crear regiones de propuesta, o lo que es lo mismo, localizar el objeto de interés, y la segunda red se encarga de la clasificación de cada región proporcionada por la red anterior. En la Figura 3.1 se ve un esquema del modelo completo.

La red neuronal base *Inception*, está compuesta por 467 capas, cuya función es extraer las características de la imagen de entrada. La mayoría de ellas sigue el esquema: 6 x *conv2d* - *merge* - *max_pooling2d* - *conv2d* - *merge* - *batch_normalization* - *activation*. Este esquema se repite 43 veces, también se añaden otras capas como *lambda*, *average_pooling2d*, *dropout*, *flatten* y *dense*. La información completa acerca de esta red se encuentra en [Maj19].

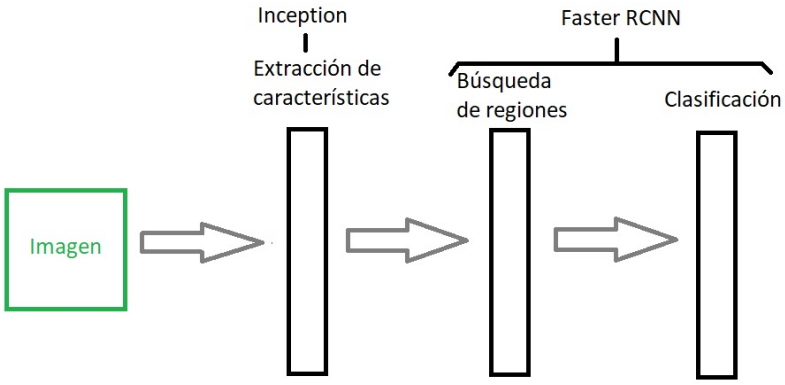


Figura 3.1: Modelo completo

Además del modelo, que se explica en el apartado 2.11.2.3, es muy importante configurarlo para adaptarlo lo mejor posible a la detección del objeto que se desee. La configuración de los modelos puede variar, y es interesante hacerlo pues dependiendo del tipo de objetos a detectar, unos parámetros son mejores que otros. En este caso, que se quiere detectar pistolas, se van a buscar los valores para los parámetros de la configuración que mejor se adapten al programa.

3.2.1. Keep Aspect Ratio Resizer

Este parámetro sirve para el cambio de tamaño en las imagenes. Para que las imagenes resultantes siempre tengan la misma relación de aspecto. Con **min_dimension** se especifica el tamaño del borde más pequeño permitido y con **max_dimension** el máximo permitido. El dataset UCM - Pistolas se compone generalmente por imágenes de 848 x 480. Si se quiere ajustar estas imágenes con altura 600, se tiene que calcular la dimension máxima así: $\frac{848*600}{480} = 1060$. Por lo que **max_dimension** tendrá el valor 1060.

3.2.2. Feature Extractor

Con este parámetro de configuración se especifica en primer lugar qué tipo de extractor o red neuronal base se escogerá para el modelo mediante **type**.

El parámetro configurable se trata del first_stage_features_stride, que significa cuánto va a profundizar, hablando en términos de convolución, para extraer las características. Por las comparativas realizadas 4.1, se decidió usar la red neuronal base Inception [SVI+16], que es un red formada por varias capas convolucionales consecutivas de 55, 33 y max-pool de forma continuada. Se necesita que la red profundice en términos de convolución para extraer las características por lo que **first_stage_features_stride** se estableció en 16.

Se ha optado por el optimizador **l2**, ya que disminuye los coeficientes y esto minimiza el efecto de correlación de entrada y hace que el modelo generalice mejor. Esto es muy importante en un detector de armas dada la diversidad de escenarios en el que va a ser expuesto.

3.2.3. First Stage Anchor Generator

Los cuadros de anclaje se utilizan en modelos de detección para ayudar a identificar objetos de diferentes formas. Para ello mediante **grid_anchor_generator** se puede modificar opciones como:

Scales: se definen para usar un conjunto de escalas explícitamente definido.

Aspect ratios: Relación de aspectos para los cuadros de anclajes en cada punto de rejilla. Esto es un atributo de proyección de imagen que describe la relación proporcional entre el ancho de una imagen y su altura.

Height stride: La altura de pixeles para cada cuadro de anclaje.

Width stride: La anchura de píxeles para cada cuadro de anclaje.

3.2.4. Initializer

Se utiliza el inicializador **truncated_normal_initializer** ya que genera números aleatorios cuya media es cercana al 0. El valor de **first_stage_max_proposals** se ajusta, ya que no se quiere que haya demasiadas propuestas a evaluar en el modelo. Por tanto se asigna el valor 150. No se utiliza dropout ya que la red no esta en riesgo de sobrealimentación.

Se ajustan los valores **max_detections_per_class** y **max_total_detections** por tener sólo una clase y no se quiere que haya un sobreprocesamiento. Se pone, por tanto, los valores 50 y 150 respectivamente. Se utiliza la función **Sigmoide** ya que Softmax es utilizado para clasificaciones multiclase.

3.2.5. Etapas

Faster RCNN se compone de dos redes, la primera propone regiones en las cuales se puede encontrar objetos (RPN). La segunda red intenta detectar objetos en las propuestas dadas por la primera. Por convención a la primera red se le denomina first stage y a la segunda second stage.

3.2.5.1. Primera Etapa

Parámetros para la primera etapa:

- **first_stage_nms_score_threshold**: El umbral de puntución de no supresión máxima.
- **first_stage_nms_iou_threshold**: El umbral de IOU sin supresión máxima.
- **first_stage_max_proposals**: El máximo de propuestas permitidas para la primera red.
- **first_stage_localization_loss_weight**: Factor de Perdida de peso por localización.
- **first_stage_objectness_loss_weight**: Factor de pérdida de peso objetivo.

El umbral de no supresión máxima se utiliza para evitar que los cuadros de anclaje delimitadores se suporpongan señalando el mismo objeto. Para que no existan varias detecciones sobre un mismo objetivo.

El índice de Jaccard mide el grado de similitud entre dos conjuntos.

$$T = \frac{Nc}{Na + Nb + Nc}$$

$$Na = Elementos_A$$

$$Nb = Elementos_B$$

$$Nc = Elementos_{Interceptados}$$

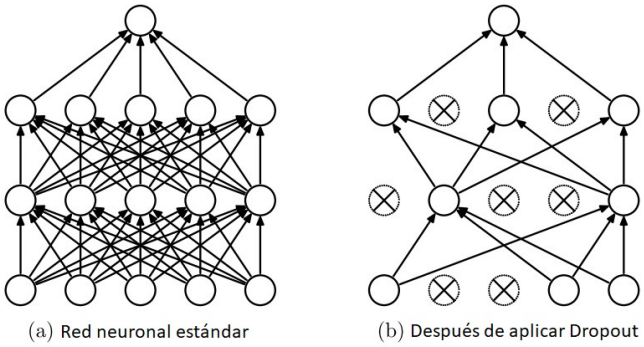
Umbral IOU: durante el entrenamiento se procede a juntar el cuadro real con el predicho sobre intersección sobre unión que es el índice de Jaccard. Los mejores cuadros se etiquetaran como positivos si están por encima del umbral IOU.

3.2.5.2. Segunda Etapa

mask_rcnn_box_predictor Se encarga de predecir clases; opcionalmente permite la predicción de máscaras o puntos clave dentro de las cajas de detección.

Los parámetros de configuración del mismo son los siguientes:

use_dropout: Generalmente se utiliza cuando la red esta en riesgo de sobrealimentación, es decir cuando la red es demasiado grande, entrenas durante mucho tiempo o si no tienes suficientes datos. Como se ve en la Figura 3.2, se desactivan neuronas de forma aleatoria.



(a) Red neuronal estándar (b) Después de aplicar Dropout

Figura 3.2: Dropout

Se recomienda no utilizar el dropout, debido a las relaciones codificadas en los mapas de características, las activaciones pueden ser altamente correlacionadas. Actualmente se utiliza batch normalization para estabilizar la red neuronal.

dropout_keep_probability: Se utiliza para calcular si la neurona tendra deserción o no, es decir, se calcula la contribución de cada neurona con la probabilidad que se indica en este parámetro.

Regulizer: Cuando se utilizan regularizadores, éstos actuan de manera que evitan el sobreajuste (overfitting), suavizando los resultados. Evitando que la máquina intente adaptarse lo máximo posible a los datos de entranamiento, ya que no da abstracción a sus predicciones, se tienen que conseguir resultados lo más genéricos posibles. Están disponibles los siguientes:

- **L1_regularizer** Agrega el valor absoluto del coeficiente de peso como término a la función de perdida.
- **L2_regularizer** Agrega el cuadrado del coeficiente de peso como término de penalización en la función de pérdida.

La función de pérdida mide con los resultados de las predicciones y la respuesta correcta que tan buenas son las predicciones. Existen varias funciones de perdida como el error cuadrático medio o la entropía cruzada.

El error cuadrático medio MSE, es la media de la diferencia entre los puntos reales y la salida predicha al cuadrado. Este método penaliza las diferencias mayores.

Initializer
Al inicializar una red profunda, puede resultar favorable mantener constante la escala de la varianza de entrada, para que no disminuya al alcanzar la capa final.

Variance scaling initializer
Este inicializador está diseñado para mantener la escala de los gradientes aproximadamente al mismo valor durante todas las capas.
Pesos
Los pesos son inicializados, la red implementa una serie de transformaciones que son aleatorias. Por ello se tendrá en la función de pérdida unos valores muy altos. A medida que la red va procesando nuevos casos esta se va ajustando. Para la segunda capa, se puede indicar el factor de pérdida de peso por localización y por clasificación mediante *second_stage_localization* y *second_stage_classification_loss_weight*.

3.2.6. ROI Polling (Regiones de Interés)

Esta capa forma parte de la red neuronal, por lo que permite reutilizar el mapa de características de la red convolucional, con esto se logra una aceleración importante en el entrenamiento, ya que se tiene una forma abstracta de representación que reduce el número de parámetros a aprender.

Initial crop Size: Corte ROI basado en la interpolación bilineal, esta es una técnica para calcular valores de una ubicación de una malla basada en celdas de cuadrículas cercanas. Se usa un promedio de distancia para estimar celdas más cercanas a las que se les dan pesos más altos.
Maxpool Kernel size: Se trata de la dimensión del núcleo de la capa de agrupación, se recomienda que esta no tiene que ser demasiada grande ya que se pierde información o características imporantes.
Maxpool Stride: Paso de la operación de grupo máximo durante la agrupación ROI

3.2.7. Optimizer

Hay diferentes tipos de optimizadores disponibles **RMSPropOptimizer**, **MomentumOptimizer** y **AdamOptimizer**.

En este caso resultó interesante el optimizador **Momentum**, ya que ayuda a acelerar el descenso de gradiente en la dirección relevante y quita repercusión a las oscilaciones. Se puede modificar el parámetro del ratio de aprendizaje mediante la configuración de **manual_step_learning_rate**, también se utiliza el valor del optimizador de momentum **momentum_optimizer_value** que se suele establecer en 0.9.

use_moving_average: La media móvil se utiliza para suavizar los datos creando promedios actualizados constantemente.

Se ajustan los parámetros del ratio de aprendizaje según el programa de entrenamiento.

```
learning_rate: {
    manual_step_learning_rate {
      initial_learning_rate: 0.0002
      schedule {
        step: 20000
        learning_rate: .0002
      }
      schedule {
        step: 40000
        learning_rate: .000002
      }
    }
  }
```

3.2.8. Otros Parámetros

Se utiliza el modelo pre-entrenado de coco para facilitar el trabajo y que la red no parta de 0, ya que como se demuestra en muchos trabajos como [MDES16], el uso de redes preentrenadas mejora el entrenamiento del modelo.

Se considera que con **num_steps** = 70000 el clasificador ya puede dar buenos resultados. Esto es debido a que un menor número de pasos sería insuficiente para un buen entrenamiento, y un mayor número de pasos conlleva un tiempo de entrenamiento demasiado alto para la capacidad computacional que se tiene.

gradient_clipping_by_norm El recorte de gradiente permite evitar acantilados, estos ocurren comúnmente en redes recurrentes en el área donde la red se comporta de manera lineal como se observa en la Figura 3.3. Con el recorte se hace que el gradiente descienda al mínimo.

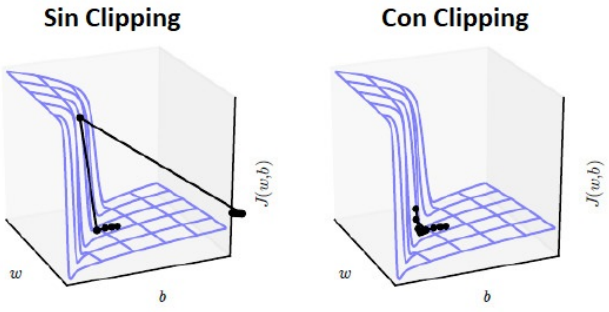


Figura 3.3: Clipping

fine_tune_checkpoint: Se tiene que indicar aquí el *path* del modelo pre-entrenado para continuar su entrenamiento. Se puede omitir pero la red empezaria de 0.
from_detection_checkpoint: Si es falso, asume que el punto de control era de un punto de control de clasificación de objetos. Es mejor empezar en un punto de control de detección que de clasificación, ya que se tendrá una capacitación más rápida.

3.2.9. Entrada de Evaluación del Entrenamiento

En ambos casos se necesita indicar el *path*, donde se encontrará el *tf record* de los datos de entranemiento y pruebas. Además del mapa de etiquetado, que indica cuántas clases tendrá el clasificador.

3.2.10. Configuración de la Evaluación

En este apartado se tendrá que indicar el número de elementos que compondrán la carpeta de test, y que, por lo tanto, van a ser evaluados.
La colección se compone de 577 imágenes de prueba, por lo que se asigna ese valor a **num_examples**. Se limitan las evaluaciones a 5 al no tener una colección grande, asignando dicho valor a **max_evals**.

Capítulo 4

Experimentación y Comparativa

Para llevar a cabo la experimentación con los modelos, y ver cuál es el que da los mejores resultados para el programa, se ejecutan los modelos elegidos en un entorno con las mismas características y usando la misma colección de datos para todos ellos. Con esto se crea un *benchmark* que hará posible ver las diferencias entre los modelos sin que otros factores afecten a los resultados.

Las características del benchmark son las siguientes:

- **CPU**: Procesador Intel Core i7-8750H CPU 2.21GHz
- **Memoria RAM**: 16Gb
- **Tarjeta gráfica**: NVIDIA GeForce GTX 1050
- **Tarjeta gráfica**: NVIDIA GeForce GTX 1050

El tiempo de entrenamiento en cada modelo probado es de 1 día. Para comparar los resultados se tienen en cuenta los siguientes factores:

- **Precisión**: Porcentaje que muestra cuántas veces la imagen contiene una pistola cuando el programa dice que hay una.
- **Efectividad**: Porcentaje que muestra cuántas pistolas detecta el programa del total de pistolas.
- **Exactitud**: Porcentaje que muestra cuántas veces acierta el programa con su diagnóstico.
- **Verdadero positivo**: Número de veces que el programa detecta acertadamente una pistola.
- **Verdadero Negativo**: Número de veces que el programa detecta acertadamente la ausencia de una pistola.
- **Falso Positivo**: Número de veces que el programa detecta una pistola, pero no hay pistola en la imagen.
- **Falso Negativo**: Número de veces que el programa no detecta una imagen de pistola, pero hay una pistola.
- **FPS**: Muestra el número de imágenes que el programa puede procesar cada segundo.

$$Precisión = \frac{VP}{VP + FP}$$

$$Efectividad = \frac{VP}{VP + FN}$$

$$Exactitud = \frac{VP + VN}{VP + FP + VN + FN}$$

4.1. Experimento 1

Se decide empezar a implementar un detector de armas usando una API de Tensorflow. Para ello se hacen 4 implementaciones distintas en las que se usa la combinación de 2 modelos distintos con 2 redes neuronales base.
Las combinaciones que se han usado son:

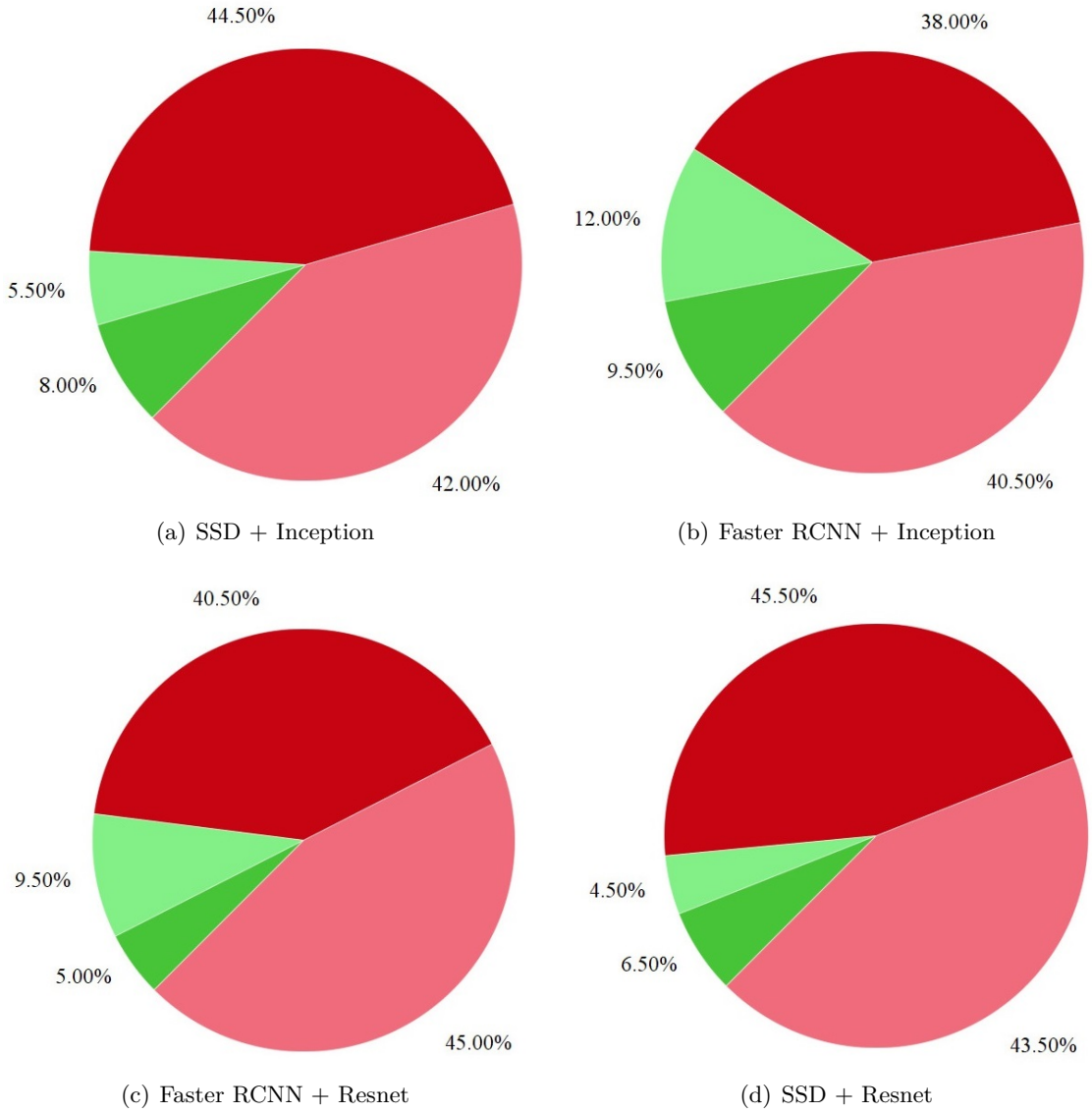
- SSD + Inception.
- Faster RCNN + Inception.
- Faster RCNN + Resnet.
- SSD + Resnet.

Se decide no usar el modelo YOLO porque, a pesar de tener buenos resultados en general, los resultados que se obtienen para objetos de pequeño tamaño es de los peores. Teniendo en cuenta que las pistolas que aparecen en los vídeos son de pequeño/medio tamaño, este modelo no es el más adecuado para el programa.

También se descartan otros modelos porque no están suficientemente probados, y por lo tanto no hay una buena documentación respecto a los mismos. Un ejemplo es *RefineDet*.

Se descartan otros modelos porque, a pesar de tener una gran precisión, necesitaban tal cantidad de procesamiento que no era viable usarlos con un procesador de ordenador normal.

Se realiza el entrenamiento de los 4 modelos con una misma colección de datos base que contiene 1000 imágenes de pistolas, para que los resultados obtenidos sean equiparables. Como se observa en la Tabla 4.1 y en la Figura 4.1, los resultados obtenidos son más positivos para Faster RCNN + Inception. Los 2 modelos cuya red neuronal base es Inception obtienen mejores resultados que los que usan Resnet. Faster RCNN con Inception obtiene mejores resultados que SSD con Inception, por lo que se decide que lo más apropiado para la detección de pistolas es **Faster RCNN + Inception**.



(a) SSD + Inception

(b) Faster RCNN + Inception

(c) Faster RCNN + Resnet

(d) SSD + Resnet

Figura 4.1: Experimento 1

Comparación 4.1: Experimento 1

	SSD + Inception	**Faster RCNN + Inception**	**Faster RCNN + Resnet**	**SSD + Resnet**
Precisión	15 %	20 %	11 %	13 %
Efectividad	16 %	19 %	10 %	13 %
Exactitud	13 %	21 %	14 %	11 %
Verdadero Positivo	16	19	10	13
Verdadero Negativo	11	24	19	9
Falso Positivo	89	76	81	91
Falso Negativo	84	81	90	87
FPS	2,1	0,5	0,4	1,8

4.2. Experimento 2

El objetivo de este experimento es conseguir una colección de imágenes que proporcione los mejores resultados posibles. por ello, a lo largo del experimento se irán haciendo mejoras continuas sobre cada una de las colecciones de imágenes, mostrando en cada mejora los resultados que se obtienen.
Se usa el modelo Faster RCNN + Inception en todos los experimentos en adelante debido a que proporciona los mejores resultados, como se demuestra en el anterior experimento 4.1.

4.2.1. Faster RCNN con Primera Colección de Datos

En internet hay varias colecciones de imágenes de armas. Éstas contienen imágenes de varios tipos de armas en posiciones variadas, pero en la mayoría de veces no están siendo usadas por nadie y simplemente están expuestas.
La primera colección de imágenes que se decide hacer consiste en una colección de imágenes [dG19] de 3000 armas que se usa en muchos programas de detección de armas y una colección de imágenes de cuchillos de 1000 imágenes.

4.2.2. Faster RCNN con Segunda Colección de Datos

Los resultados de la colección de datos 4.2.1 son muy malos, tal como se puede observar en la Tabla 4.2 debido a que las imágenes de la colección de imágenes no se asemejan a las imágenes de los vídeos que se usan para hacer las detecciones (vídeos de cámaras de videovigilancia o parecido).
Estos resultados demuestran que había que incluir imágenes en las que aparezcan las pistolas siendo usadas. Para ello se coge otra colección de imágenes de internet que contiene 120 imágenes que tienen relación con el tipo de vídeos que van a ser usados por el programa. Se deja de usar la clase de cuchillos, y se usan exclusivamente imágenes de una sola clase para optimizar el entrenamiento del modelo.

4.2.3. Faster RCNN con Tercera Colección de Datos

Tras comprobar los resultados se decide hacer una colección de imágenes casera de pistolas cogiendo los fotogramas de 3 vídeos, y con un programa llamado “OpenLabeling” [Car19] se decide hacer una por una las etiquetas de todos los fotogramas de los 3 vídeos. Se junta la nueva colección de imágenes con la colección de imágenes anterior, y se decide entrenar de nuevo el modelo.

4.2.4. Faster RCNN con Cuarta Colección de Datos

Los resultados de la colección de imágenes 4.2.3 son mejores que los de la colección 4.2.2, tal y como se observa en la Tabla 4.2, pero siguen sin ser buenos por las siguientes razones:

- La cantidad de imágenes que se cogen de los vídeos es mucho mayor a la que se tiene en las otras colecciones de imágenes, por lo que la colección de imágenes está desproporcionada.
- Al usar sólo 3 vídeos no se consigue entrenar el modelo para una variedad amplia de situaciones, y sólo se hacen detecciones en situaciones cuyo contexto es muy parecido al de los vídeos.
- Se cogen todos los fotogramas de los vídeos, y al haber muchos fotogramas por segundo, la diferencia entre fotogramas es mínima, y no aporta información útil tener 15 fotogramas casi iguales.

Teniendo toda esta información se decide hacer una colección de imágenes, que consiste de:

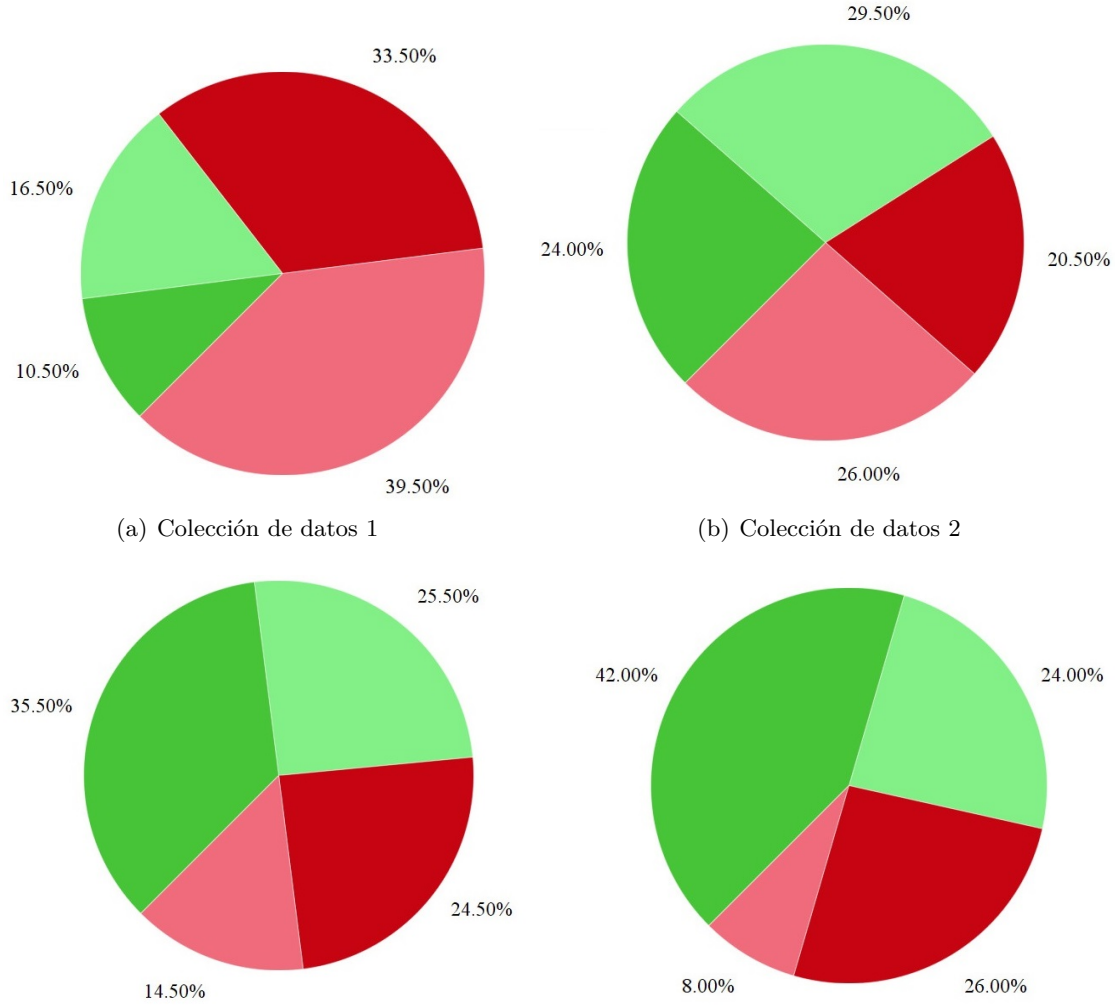
- La primera colección de imágenes (sólo las pistolas) se somete a un filtrado de imágenes una por una y se eliminan las imágenes que se considera que no tienen relevancia y no aportan beneficios al entrenamiento. (1200 imágenes).
- La segunda colección de imágenes.
- Las imágenes de los 3 vídeos se someten al mismo filtrado que el primero, para evitar imágenes repetidas que afecten al entrenamiento.
- Se seleccionan 30 nuevos videos de los cuales se cogen los 6 más relevantes y se toman las imágenes más relevantes y se etiquetan.

4.2.5. Comparativa

Como se observa en la Tabla 4.2 y en la Figura 4.2, cada colección de imágenes afecta a la eficacia del modelo. Sin duda, la mejor colección de imágenes es la última, que es la que se decide usar para hacer el siguiente experimento 4.3.

Comparación 4.2: Experimento 2

	Colección 1	**Colección 2**	**Colección 3**	**Colección 4**
Precisión	24 %	54 %	59 %	61 %
Efectividad	21 %	48 %	71 %	84 %
Exactitud	27 %	53 %	61 %	66 %
Verdadero Positivo	21	48	71	84
Verdadero Negativo	33	59	51	48
Falso Positivo	67	41	49	52
Falso Negativo	79	52	29	16
FPS	0,5	0,5	0,5	0,5



(a) Colección de datos 1

(b) Colección de datos 2

(c) Colección de datos 3

(d) Colección de datos 4

Figura 4.2: Experimento 2

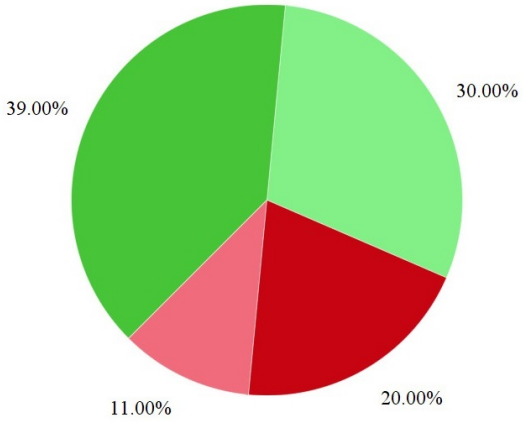
4.3. Experimento 3

En este experimento se llevan a cabo cambios en la configuración del modelo. Se modifican los parámetros que afectan al modelo, tal y como se explica en el apartado 3.2.

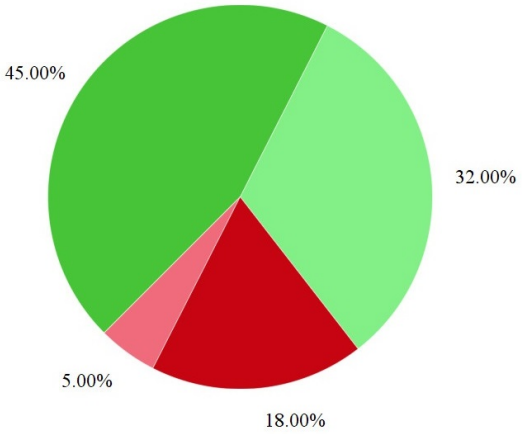
Este experimento consta de tres parte. Primero se realizan 3 configuraciones distintas 4.3.1, partiendo de la predeterminada que ofrece el modelo. Después se realizan 3 nuevas configuraciones 4.3.2 partiendo de la mejor configuración de la primera parte. Por último, se realiza una mejora 4.3.3 a la colección de datos, y se usa la mejor configuración.

4.3.1. Primeras Configuraciones

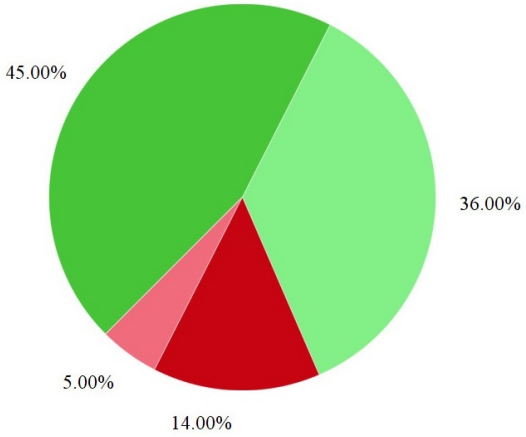
Se aplican ligeros cambios en la configuración del modelo, para así intentar mejorar los resultados que se obtienen en la detección de pistolas. Es posible que una configuración sea buena para la detección de un tipo de objetos, pero no para la detección de otros. Es por esto que se intentan pequeños cambios para ver de qué forma mejora la detección aplicando los cambios.



(a) Configuración 1



(b) Configuración 2



(c) Configuración 3

Figura 4.3: Experimento 3a

Comparación 4.3: Experimento 3a

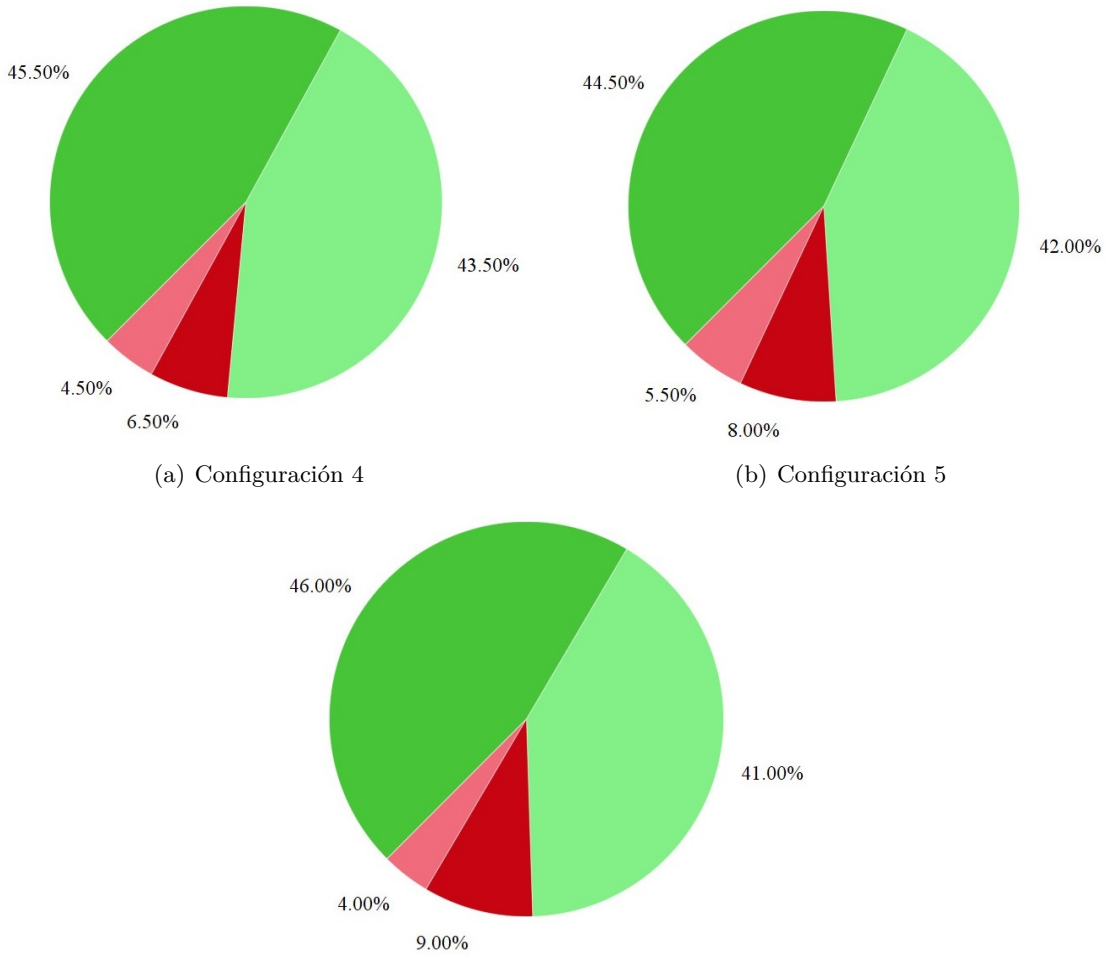
	Configuración 1	Configuración 2	Configuración 3
Precisión	66 %	71 %	76 %
Efectividad	78 %	90 %	90 %
Efectividad	69 %	77 %	81 %
Verdadero Positivo	78	90	90
Verdadero Negativo	60	64	72
Falso Positivo	40	36	28
Falso Negativo	22	10	10
FPS	0,5	0,5	0,5

4.3.2. Segundas Configuraciones

En esta segunda parte del experimento 3, se analizan los resultados de la primera parte del experimento, y se deciden reajustar los parámetros de la configuración 3, para así obtener mejores resultados. Como se observa en la Tabla 4.4 y en la Figura 4.4, la configuración 4 es la que mejores resultados proporciona se puede ver explicada en el apartado 3.2.

Comparación 4.4: Experimento 3b

	Configuración 4	Configuración 5	Configuración 6
Precisión	87 %	85 %	84 %
Efectividad	91 %	89 %	92 %
Exactitud	89 %	89 %	87 %
Verdadero Positivo	91	89	92
Verdadero Negativo	87	84	82
Falso Positivo	13	16	18
Falso Negativo	9	11	8
FPS	0,5	0,5	0,5



(a) Configuración 4

(b) Configuración 5

(c) Configuración 6

Figura 4.4: Experimento 3b

4.3.3. Última Colección de Imágenes

El último experimento es una pequeña mejora en la colección de datos, que consiste en igualar el número de imágenes en las que aparecen pistolas y en las que no. Esto se hace porque se está sobreentrenando a la red para que diga que hay una pistola más veces de las que debería. Esto provoca un gran número de FP , que como bien se observa en la Tabla 4.5 y en la Figura 4.5, se ve reducido tras la implementación de la quinta colección de imágenes.

Tabla 4.5: Experimento 3c

Parámetro	Valor
Precisión	90 %
Efectividad	92 %
Exactitud	91 %
Verdadero Positivo	92
Verdadero Negativo	90
Falso Positivo	10
Falso Negativo	8
FPS	0,5

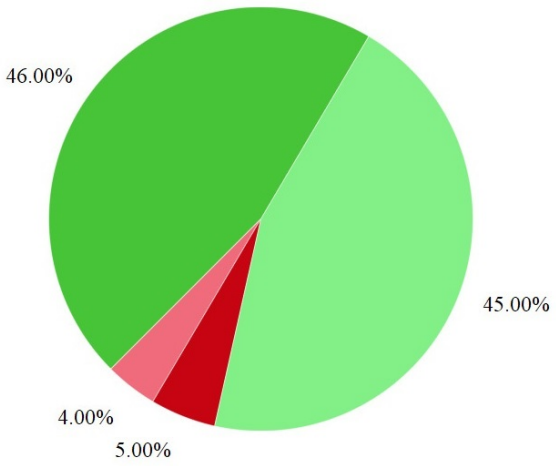
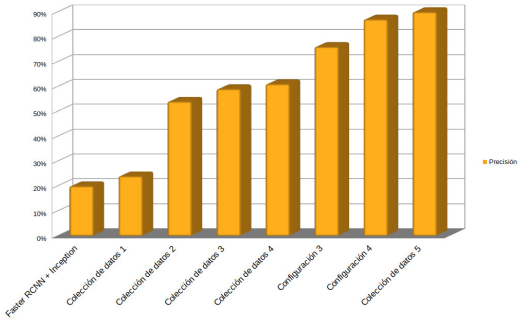


Figura 4.5: Experimento 3c

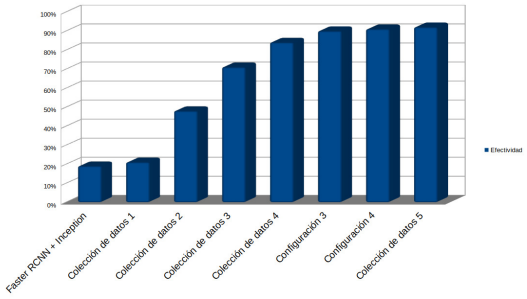
4.4. Resultados

Los resultados finales son prometedores. Tras haber hecho los experimentos finales con los modelos elegidos 4.1, y viendo que el modelo Faster RCNN + Inception es el que mejores resultados ha dado. Se presentan los resultados del modelo y el dataset finales, que son los que mejor hacen la detección:

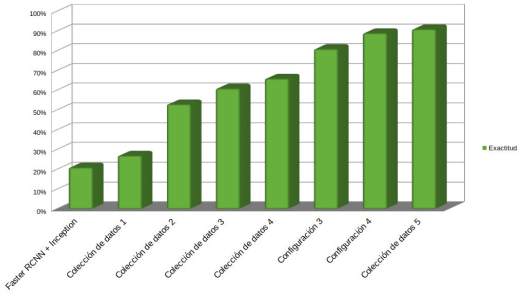
Los resultados de los experimentos son muy buenos. A continuación se muestran los resultados de las distintas mejoras que se han ido haciendo en los experimentos:



(a) Precisión



(b) Efectividad



(c) Exactitud

Figura 4.6: Resumen experimentos

Como se puede observar en la Figura 4.6, con cada paso en los experimentos, se han ido mejorando la precisión, efectividad y exactitud del programa, llegando a una precisión del 90 %, efectividad del 92 % y exactitud del 91 %.

4.4.1. Imágenes de Ejemplo

A continuación se muestran imágenes que se han pasado por el programa:

Figura 4.7: Antes y depués 1

Figura 4.8: Antes y depués 2

Figura 4.9: Antes y depués 3

Figura 4.10: Antes y depués 4

Figura 4.11: Antes y depués 5

Figura 4.12: Antes y depués 6

Capítulo 5

Conclusiones y Trabajo Futuro

5.1. Conclusiones

El programa final sin duda consigue hacer una detección aceptable de armas en vídeos. Se ha aprendido mucho en el proceso de creación del programa, y se han dado pasos más grandes a medida que se avanzaba en su desarrollo.

En un principio se tenía una buena arquitectura en el modelo 4.1, pero los resultados no la acompañaban. Esto es debido a que es imprescindible tener una buena colección de imágenes para que los resultados sean buenos, tal y como se ha visto en el experimento 4.2.

La relevancia de las imágenes contenidas en la colección de imágenes es muy importante. Se tardó mucho en conseguir una buena colección, ya que se precisa no sólo de mucha cantidad de imágenes, sino de calidad en estas. Esto se vió demostrado en el experimento 4.3.3.

Se pensaba que se podría conseguir buena precisión en el detector y conseguir que fuera rápido también, pero con un procesador normal no se pueden conseguir las dos cosas, tal y como se demuestra en el experimento 4.1.

El modelo no sólo es la arquitectura de las capas que lo forman, sino también la configuración que posee. Se ha demostrado que modificando la configuración del modelo, se obtienen mejores resultados en el detector.

Se puede encontrar el código fuente del programa en el siguiente enlace: https://github.com/Alejandromndza/UCM-Tensorflow [EM19].

5.2. Trabajo Futuro

Debido al largo tiempo que constituye entrenar un modelo, 1 día en nuestro procesador, no se han podido probar todas las mejoras que se qerían. Por lo que sería recomendable hacer uso de un procesador mejor con GPU. Se podría probar a usar un modelo distinto, ya que si se usa una buena GPU se pueden obtener mejores resultados con otro modelo.

Las configuraciones que se han probado no son todas las posibles. Se ha intentado hacer pequeñas modificaciones en el archivo de configuración, e ir midiendo los resultados para saber qué había que cambiar y qué valor había que darle a cada parámetro para optimizar los resultados del programa, así como para disminuir el alto tiempo de entrenamiento, que ha sido sin duda uno de los mayores problemas que se ha tenido. Sin embargo, se podría mejorar haciendo más entrenamientos y experimentos.

No se ha podido dedicar el tiempo suficiente al dataset, y un trabajo futuro sería poner mayor variedad de vídeos y mayor cantidad. También se propone hacer un preprocesamiento a las imágenes para disminuir la tasa de FP , pues es uno de los valores que más nos ha costado mejorar y que más importante se considera.

Otra mejora relacionada con el dataset sería aplicar *Data augmentation*. Apenas se ha hecho data augmentation en la colección de imágenes (se han aplicado giros aleatorios en las imágenes de entrenamiento). Una forma de mejorar el dataset sería cambiar la resolución, el brillo, la escala, la rotación, hacer zoom, aumentar la saturación, etc.

Se propone la creación de una interfaz gráfica, para facilitar al usuario el uso de la herramienta. Esta debería constar de una entrada para el vídeo, y la posibilidad de usar las clases que se quieran con tantas entradas para colecciones de imágenes como número de clases se seleccionen.

Una última mejora que se propone sería la de implementar nuevas clases. El modelo que se propone en este trabajo ya cuenta con la implementación en el modelo para hacerlo, por lo que sólo sería necesario hacer una colección de imágenes de otra clase para entrenar al modelo.

Capítulo 6

Introduction

Nowadays, there is a lot of unused information. In this case, we refer to the information contained in videos. So much digital content is generated that it is impossible to monitor it with human means. This is why the need arises for the use of artificial intelligence techniques to process information in a massive way.

In this work we are going to apply DL for the detection of handguns in videos. We will try several models, and will apply improvements on the chosen model to increase the detection.

To be able to do this we will create a dataset of images with labels indicating where the handguns are inside the image. We will do this manually due to the scarcity of handgun datasets on the internet.

6.1. Motivation

The detection of weapons in videos is a problem that increases daily, as it affects the safety of people. Although it is an important problem, there are not many programs dedicated to it. We have found some software that does this, but the datasets of images they use are scarce and irrelevant to the case.

There are also works related to the detection of knives in videos, but they are not applicable to the detection of guns.

Of course, there is still a lot of progress to be made in the field of weapon detection in videos, because with the recent object detection models, you can make a weapon detection software that is more accurate than the previous ones.

6.2. Context

This End of Degree Paper is part of a research project entitled RAMSES, approved by the European Commission within the Horizon 2020 Research and Innovation Framework Programme (H2020-FCT-2015, Innovation Action, Proposal Number: 700326) and in which the GASS Group of the Software Engineering and Artificial Intelligence Department participates, integrated in the Faculty of Computer Science of the Complutense University of Madrid (Analysis, Security and Systems Group, http://gass.ucm.es, group 910623 of the catalogue of research groups recognised by the UCM).

In addition to the Complutense University of Madrid, the following participate entities:

- Treelogic Telematics and Rational Logic for Empresa Europea SL (Spain)
- Ministério da Justiça (Portugal)
- University of Kent (United Kingdom)
- Centro Ricerche e Studi su Sicurezza e Criminalità (Italy)
- Fachhochschule fur Offentliche Verwaltung und Rechtspflege in Bayern (Germany)
- Trilateral Research & Consulting LLP (United Kingdom)
- Politecnico di Milano (Italy)
- Service Public Federal Interieur (Belgium)
- Universität des Saarlandes (Germany)
- Dirección General de Policía - Ministerio del Interior (Spain)

6.3. Objectives and approach

The main objective of this work is the creation of a program that is capable of detecting weapons within a video, framing the weapon throughout the video.

We focus on the detection of hanguns in videos through the use of artificial intelligence techniques (in our case we will use the Faster RCNN algorithm with the Inception backbone network), which implies their classification and location in the image. This has many applications such as the prevention of criminal acts by connecting our program to video-surveillance videos. In this way, a weapon could be detected in an airport camera and alert the authorities of where the suspect is, what weapon he is carrying, who he is, etc.

It can also have other applications, such as reviewing videos to see if they contain weapons in it, and in what minute of the video the weapon is seen.

We will focus on accuracy rather than speed, because we can't allow the software not to detect a gun, or the software to say wrong where there are guns.

6.4. Work schedule

Our work is divided into 4 phases that are sometimes simultaneous: Research, implementation, experimentation and documentation.

We present the activities contained in each phase in the Table 6.1:

Tabla 6.1: Working plan

Activity	**Time**	**Date**$_{Begin}$	**Date**$_{End}$
Research	87	01/10/2018	27/12/2018
Study of Python	20	01/10/2018	21/10/2018
Study of Machine Learning	15	21/10/2018	05/11/2018
Study of Deep Learning	15	05/11/2018	20/11/2018
Study of models for object detection	15	20/11/2018	05/12/2018
Reading of papers	32	25/11/2018	27/12/2018
Implementation	120	20/12/2018	19/04/2019
Initial classificator	7	20/12/2018	27/12/2018
Dataset 1	7	27/12/2018	03/01/2019
Dataset 2	7	03/01/2019	10/01/2019
Dataset 3	10	10/01/2019	20/01/2019
Dataset 4	14	20/01/2019	03/02/2019
Configurations 1	25	03/02/2019	28/02/2019
Configurations 2	25	28/02/2019	25/03/2019
Dataset 5	25	25/03/2019	19/04/2019
Experiments	128	27/12/2018	04/05/2019
Experiment 1	10	27/12/2018	06/01/2019
Experiment 2	38	06/01/2019	13/02/2019
Experiment 3	15	28/02/2019	04/05/2019
Documentation	180	20/11/2018	19/05/2019
Summaries	30	20/11/2018	20/12/2018
Gathering information	120	20/12/2018	19/04/2019
Analysis of experiments	124	06/01/2019	10/05/2019
Writing memory	30	19/04/2019	19/05/2019

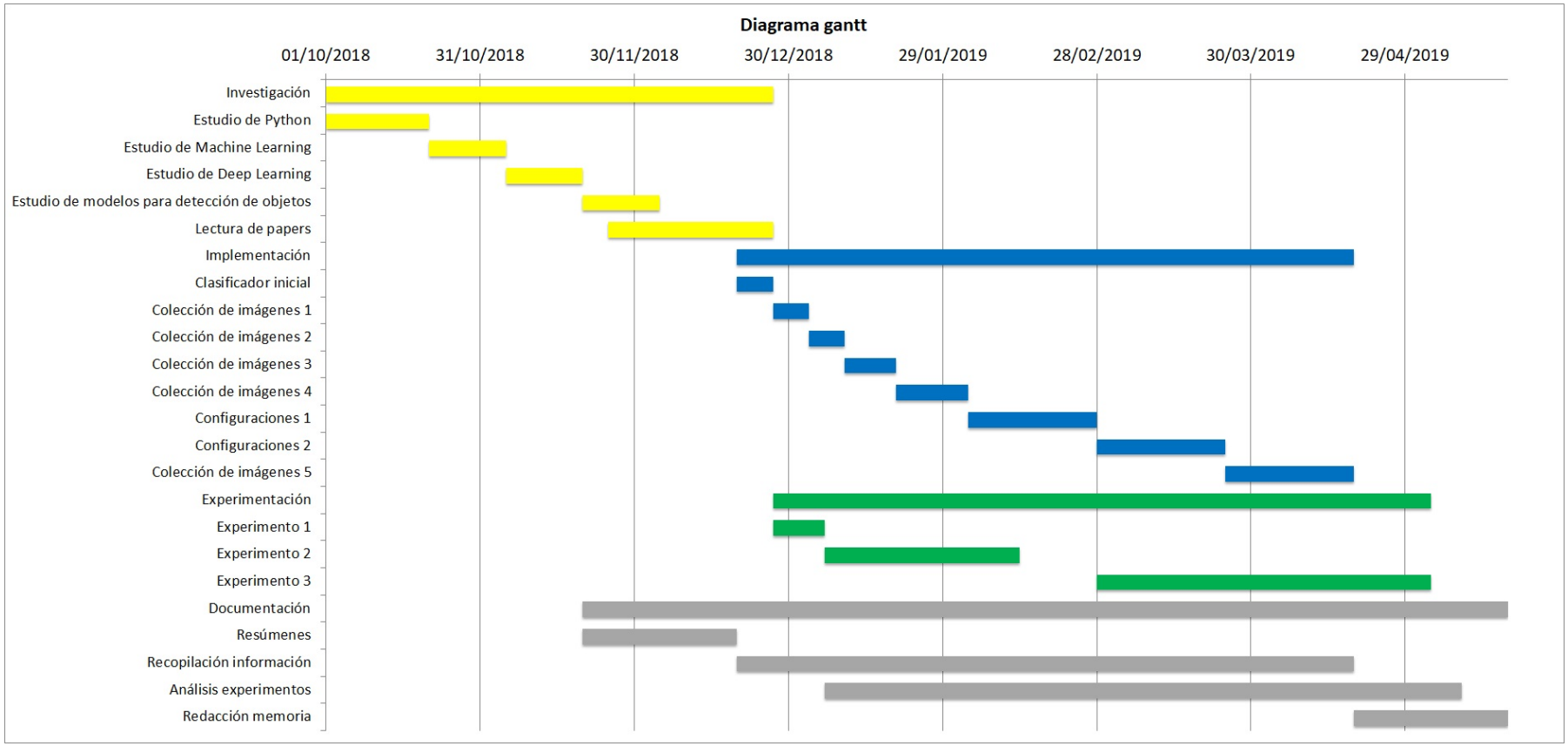


Figura 6.1: Diagrama de Gantt

6.5. Structure

The memory of the work is divided into chapters according to the phase to which they correspond as clarified below:

Chapter 1 provides a brief introduction to the project, as well as the motive behind it.

In Chapter 2 a theoretical framework of artificial intelligence is written to contextualize the project.

Chapter 3 shows the Model used in our program.

Chapter 4 is made up of several experiments in which the developments that are being made are tested and the results are analysed in order to know in which line to continue the development. It also shows the results obtained in the work.

Chapter 5 is the conclusion of the work and also discusses possible improvements it could have.

Chapter 6 is the introduction in English.

Chapter 5 is the conclusion and future work in English.

Chapter 8 is the last chapter, and describes the path taken by each of the participants in the project. It also explains the contributions that each one has made.

Capítulo 7

Conclusions and Future Work

7.1. Conclusions

The final program undoubtedly manages to make an acceptable detection of weapons in videos. Much has been learned in the process of creating the programme, and greater steps have been taken as the development of the programme has progressed.

Initially there was a good architecture in the model 4.1, but the results did not accompany it. This is because it is essential to have a good collection of images for the results to be good, as seen in the experiment 4.2.

The relevance of the images contained in the image collection is very important. It took a long time to get a good collection, as it requires not only a lot of images, but quality in these. This was demonstrated in the experiment 4.3.3.

It was thought that you could get good accuracy in the detector and make it fast as well, but with a normal processor you can't get both, as demonstrated in the experiment 4.1.

The model is not only the architecture of the layers that make it up, but also the configuration that it has. It has been shown that modifying the configuration of the model gives better results in the detector.

The source code of the programm can be found in the following link: https://github.com/Alejandromndza/UCM-Tensorflow [EM19].

7.2. Future Work

Due to the long time it takes to train a model, 1 day in our processor, we could not test all the improvements you would want. So it would be advisable to use a better processor with GPU. You could try using a different model, because if you use a good GPU you can get better results with another model.

The configurations that have been tested are not all possible. We have tried to make small changes in the configuration file, and measure the results to know what had to change and what value had to be given to each parameter to optimize the results of the program, as well as to reduce the high training time, which has been undoubtedly one of the biggest problems that has been had. However, it could be improved by doing more training and experiments.

We have not been able to dedicate enough time to dataset, and a future job would be to put more variety of videos and more quantity. It is also proposed to pre-process the images to reduce the rate of false positives, as it is one of the values that has cost us the most to improve and that is considered the most important.

Another improvement related to dataset would be to apply textData augmentation. Hardly any data augmentation has been done in the image collection (random twists have been applied in the training images). One way to improve the dataset would be to change the resolution, brightness, scale, rotation, zoom, increase saturation, etc.

It is proposed to create a graphical interface, to facilitate the user's use of the tool. This should consist of an entry for the video, and the possibility of using the classes you want with as many entries for collections of images as number of classes are selected.

A final proposed improvement would be to implement new classes. The model proposed in this work already has the implementation in the model to do so, so it would only be necessary to make a collection of images of another class to train the model.

Capítulo 8

Aportaciones Individuales

8.1. Pablo Esteve Calzado

Mi primera tarea fue aprender Python, ya que no tenía conocimientos previos en este lenguaje, e iba a necesitar aprenderlo para poder desarrollar e implementar programas basados en IA. Para aprender este lenguaje hice uso de una plataforma web llamada *Udemy*. Esta plataforma me permitió iniciarme en el lenguaje con ejemplos prácticos.

Tras esta primera aproximación con el lenguaje me vi obligado a aprender los principios básicos del AA 2.5.1 y del AP 2.5.2, ya que en nuestro proyecto íbamos a usar estos métodos para la detección 2.9. Hice uso de la misma plataforma que utilicé para el estudio de Python.

Continué con el estudio de visión artificial 2.7, para lo cual nuestros tutores nos facilitaron un libro [PyI19] que me permitió entender cómo funcionaba el AP con visión artificial.

Cabe destacar que mi compañero y yo nos reuníamos una vez a la semana para comparar nuestros avances y discutir lo que habíamos aprendido.

Hice, junto a mi compañero, un programa que clasificaba imágenes. Consideramos que la detección era un problema de mayor envergadura, y el clasificador nos permitiría aprender conceptos más básicos. Para el correcto funcionamiento de este programa me encargué de las conexiones de la colección de imágenes con el modelo. El clasificador constaba de una colección de imágenes *Cifar*, la cual contiene 10 clases y un total de 60.000 imágenes, distribuidas en 10.000 para test y 50.000 para entrenamiento.

Tras la implementación de este clasificador, mi compañero y yo pusimos en común cómo habíamos hecho nuestra parte, para que ambos entendiéramos cómo funcionaba el clasificador.

Para adentrarme en la detección de objetos en vídeos, me dispuse a leer muchos papers científicos, algunos proporcionados por nuestros tutores y otros que encontraba en google. De manera transversal, fui haciendo resúmenes de estos papers para entenderlos y tener una idea general sobre los modelos que se usan en el estado del arte de la detección. Cabe destacar que la gran mayoría de la información que se encuentra es en inglés. Esto me dificultó, al principio, comprender la terminología que se usaba en este campo.

En este momento estábamos teniendo problemas con la velocidad de cómputo, por lo que decidí cambiar el ordenador para hacer los entrenamientos de los modelos en otro ordenador con tarjeta gráfica dedicada. Finalmente, mi compañero tuvo la idea de ejecutar el código en la nube, pero no pudimos seguir haciéndolo mucho tiempo porque era de pago.

Mi compañero y yo nos dividimos para buscar implementaciones de modelos para entrenarlos con colecciones de imágenes personalizadas. Estuve buscando durante varios días, probando varios modelos e implementaciones. Uno de los modelos que implementé fue un detector de caras hecho con YOLO a través de la webcam. Al final decidimos usar una API de Tensorflow que había encontrado mi compañero. Teníamos que elegir qué modelos íbamos a probar en la API, por lo que hice acopio de la investigación realizada anteriormente y la amplié un poco más con modelos que eran del estado del arte. Decidí que debíamos usar el modelo Faster RCNN 2.11.2.3 y el SSD 2.11.1.1 ya que son los más probados, y los que mejores resultados han dado en otros experimentos.

Seguidamente, hice un estudio sobre cómo tiene que ser una colección de imágenes para que sea efectiva. Después empecé a buscar colecciones de imágenes en internet, pero no había casi ninguna colección aceptable. Me puse en contacto con el Departamento de Inteligencia Artificial de la Universidad de Granada, ya que habían realizado un software de detección de armas también, pero la colección de imágenes que usaban ya la teníamos. Empecé a recopilar imágenes y vídeos de internet en los que aparecieran pistolas, y se los pasaba a mi compañero para que hiciera el etiquetado de los mismos.

Tras hacer varios experimentos 4.2 con las distintas colecciones de pistolas pasé a estudiar cómo modificar la configuración del modelo. Hay muchos parámetros a configurar, por lo que los dividimos entre ambos para estudiarlos y, más tarde, ponerlos en común. Después de estudiarlos, hice 2 ejemplos de modificación de la configuración para el siguiente experimento 4.3.1.

Cuando finalizó el experimento, decidí realizar uno nuevo 4.3.2, donde hice una nueva configuración de los parámetros del modelo respecto al anterior experimento 4.3.1. En este experimento ya conseguimos una configuración aceptable, por lo que decidimos hacer una nueva mejora sobre la colección de datos que teníamos.

Para hacer dicha mejora me grabé en varios vídeos usando una pistola de plástico en varios escenarios. Estos vídeos se los pasé a mi compañero para que hiciera el etiquetado.

Los resultados con esta última colección de imágenes fueron muy buenos 4.3.3, por lo que empezamos a redactar la memoria usando toda la documentación que habíamos ido acumulando en todos los experimentos. Gracias a una plataforma en línea de LaTex que nuestros tutores nos proporcionaron, pude hacer la memoria a la vez que mi compañero, poniendo siempre en común los avances sobre la misma.

8.2. Alejandro Mendoza Silva

Al principio tuve que dedicar mucho tiempo al entendimiento de Python, ya que aunque tenía conocimientos previos al cursar *Bid Data* en la universidad como asignatura optativa (donde aprendí la funcionalidad básica del lenguaje), desconocía su aplicación en los campos de IA 2.5. Librerias como pandas para la manipulación de datos estructurados, numPy por su facilidad al trabajar con matrices Matplotlib para la representación de histogramas, gráficos de líneas, etc.

Entre otras muchas librerías utilizadas actualmente para el desarrollo en el ámbito de IA. Tras este estudio por encima de las características de python, realicé un estudio más intensivo para tener conocimientos más amplios. Utilice una plataforma denominada DataCamp, en la que realicé un curso de AP 2.5.2, donde explicaban conceptos como qué es una neurona, capas ocultas, función de activación, optimizadores, y así ir comprendiendo la importancia de la limpieza de datos para modelos predictivos de regresión lineal y otras problemáticas.

Esta plataforma me resultó muy interesante, y mereció la pena la aportación económica para ingresar en ella. Ya que al permitirte ejecutar el código online en el propio navegador, facilitaba el entendimiento sin tener que pelear con las librerías, entornos virtuales o versiones.

Tuve reuniones semanales con mi compañero, lo que me permitió poner a prueba mis conocimientos y discutir cómo funcionaban los métodos que habíamos aprendido. Tras varias reuniones con nuestros tutores encaminé el aprendizaje hacia nuestro objetivo, la detección en videos de pistolas 2.9. Mi compañero y yo decidimos desarrollar un clasificador, ya que el entrenamiento de un detector era más complejo porque entran en juego más factores como los archivos xml donde sitúan al objeto a detectar en la imagen. Para el desarrollo de este clasificador me encargué de configurar el modelo de la red neuronal que íbamos a usar.

Después de una larga investigación por artículos de google y leyendo papers científicos de desarrollo en AP tuve una idea de cómo funcionaban las redes neuronales convolucionales. Hay que remarcar aquí el hecho de que al ser algo complejo, hay mucha información confusa que dificultó el estudio sobre estos temas. Fue interesante contar con una investigadora de AP la cual, nos ayudó a comprender y esclarecer todas las dudas que nos habían surgido hasta el momento.

Tras las reuniones me dispuse a aprender de una manera más profunda el funcionamiento de las redes neuronales convolucionales. Keras tomó un papel fundamental desde el primer momento, ya que al ser una librería de alto nivel de tensorflow su nivel de abstracción hizo más fácil la comprensión de los modelos como el clasificador de imágenes que se desarrolló anteriormente. Además con las aclaraciones descritas anteriormente se pudo continuar su desarrollo y no se volvió algo tan tedioso. Para el diseño del modelo se tuvo como referencia arquitecturas como la VGG-16, que se basan en hacer creciente el número de neuronas de las capas convolucionales. Se obtuvieron buenos resultados modificando los parámetros del modelo hasta llegar al definitivo que es el que se encuentra en nuestro Jupyter notebook.

Como la velocidad de cómputo era un problema, mi compañero y yo buscamos alternativas a este problema, en primera instancia mi compañero cambió el portátil por uno mejor, con la ventaja de tener una tarjeta gráfica dedicada, lo que permitió el cómputo en gpu. Se obtuvieron buenos resultados, pero no lo esperado. Por lo tanto decididí hacer la ejecución en cloud, para ello Google dispone de un Jupyter notebook, que me permitió ejecutar código con una GPU Tesla de gran potencia.

Ya que nuestro objetivo era un detector y no un clasificador, se buscó la manera de entrenar modelos predefinidos como el Faster RCNN 2.11.2.3 o el modelo SSD 2.11.1.1. Tuve que realizar varios experimentos, buscar información relevante y tras varios intentos fallidos, di con la API oficial de TensorFlow donde te explican cómo entrenar objetos custom. Tras la lectura y comprensión sobre cómo funciona este API. Llevé a cabo, junto con mi compañero, un estudio para determinar qué modelos y redes neuronales base de los disponibles nos interesaban para la experimentación.

Al cabo de varios días de estudio decidí usar Inception y Resnet como redes neuronales base. Para el entrenamiento tuve que entender el funcionamiento del API y gracias a un código de github, pude entender cómo se generan los tf records necesarios para el entrenamiento.

Hicimos el primer experimento con los modelos elegidos por mi compañero y las redes neuronales base elegidas por mi. Tras los resultados del experimento decidimos hacer una colección de imágenes. En un principio me encargué yo de etiquetar imágenes mientras mi compañero buscaba las imágenes.

El proceso más laborioso ha sido hacer el etiquetado de todas las imágenes con un programa llamado *OpenLabeling* [Car19]. Sumando todas las colecciones de imágenes 3.1 he invertido mucho tiempo en hacer el etiquetado de casi 25k imágenes. De las cuales muchas se desecharon por similitud con las otras imágenes y no aportaban información relevante para la red. Después de la mejora de las colecciones de imágenes tuve un proceso de estudio de la configuración del modelo entero de Faster RCNN diviediendo con mi compañero los parámetros que había que cambiar. Tras poner en común la información sobre los parámetro, hice una mejora de la configuración para el primer experimento 4.3 con las configuraciones.

Tras este experimento hice dos nuevas configuraciones para hacer el segundo experimento 4.3.2 con las configuraciones. De este experimento sacamos la configuración final que íbamos a usar.

Hice un último etiquetado de imágenes con vídeos de mi compañero para añadirlo a la colección de imágenes 4.3.3, y añadí también imágenes en las que no aparecían armas, para equilibrar la proporción de imágenes con pistolas y sin pistolas.

La documentación de la memoria la fui haciendo a medida que avanzábamos, y la redacción final de la misma fue simplemente escribirla de forma más ordenada usando el programa LaTex. Gracias a una plataforma de LaTex en la nube proporcionada por nuestros tutores pude hacer, junto a mi compañero, la redacción de la memoria en línea.

Bibliografía

[AmAaP+18] M. Al-masni, M. A. Al-antari, J. Park, G. Gi, T. Kim, P. Rivera, E. Valarezo, M. Choi, S. Han, and T. Kim. Simultaneous Detection and Classification of Breast Masses in Digital Mammograms via a Deep Learning YOLO-based CAD System. *Computer Methods and Programs in Biomedicine*, 157, April 2018.

[Bre01] C. Breazeal. MIT team building social robot. http://news.mit.edu/2001/kismet, February 2001.

[Car19] Cartucho. OpenLabeling. https://github.com/Cartucho/OpenLabeling, May 2019.

[Com19] History Computer. Logic Theorist: History. https://history-computer.com/ModernComputer/Software/LogicTheorist.html, May 2019.

[CTP+19] A. Castillo, S. Tabik, F. Pérez, R. Olmos, and F. Herrera. Brightness guided preprocessing for automatic cold steel weapon detection in surveillance videos with deep learning. *Neurocomputing*, 330:151 – 161, 2019.

[dG19] Universidad de Granada. Weapons detection. https://sci2s.ugr.es/weapons-detection, May 2019.

[EM19] P. Esteve and A. Mendoza. Detector armas. https://github.com/Alejandromndza/UCM-Tensorflow, May 2019.

[FH99] H. Feng-Hsiung. IBM's Deep Blue Chess grandmaster chips. *IEEE Micro*, 19(2):70–81, March 1999.

[GDDM14] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Ohio, USA, June 2014.

[GN09] P. Gehler and S. Nowozin. On feature combination for multiclass object classification. In *2009 IEEE 12th International Conference on Computer Vision*, pages 221–228, Kyoto, Japan, Sep. 2009.

[HOT06] G. E. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, May 2006.

[HS95] S. Hochreiter and J. Schmidhuber. Long Short-term Memory. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.3117, August 1995.

[HS97] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November 1997.

[JLM10] V. Jain and E. Learned-Miller. FDDB: A Benchmark for Face Detection in Unconstrained Settings. Technical Report, University of Massachusetts, Amherst, June 2010.

[KPNY19] S. Kim, S. Park, Byunggook Na, and S. Yoon. Spiking-YOLO: Spiking Neural Network for Real-time Object Detection. *arXiv e-prints*, March 2019.

[KSL+19] T. Kong, F. Sun, H. Liu, Y. Jiang, and J. Shi. Consistent Optimization for Single-Shot Object Detection. Technical Report, Tsinghua University, January 2019.

[Maj19] S. Majumdar. Inception. https://github.com/titu1994/Inception-v4/blob/master/inception_resnet_v2.py, June 2019.

[MDES16] D. Marmanis, M. Datcu, T. Esch, and U. Stilla. Deep Learning Earth Observation Classification Using ImageNet Pretrained Networks. *IEEE Geoscience and Remote Sensing Letters*, 13(1):105–109, Jan 2016.

[Mon19] Monografías. Redes neuronales. https://www.monografias.com/trabajos12/redneuro/redneuro2.shtml, May 2019.

[OTH18] R. Olmos, S. Tabik, and F. Herrera. Automatic handgun detection alarm in videos using deep learning. *Neurocomputing*, 275:66 – 72, January 2018.

[PC19] F. Pérez and A. Castillo. Weapons Detection. http://sci2s.ugr.es/weapons-detection, May 2019.

[PyI19] PyImageSearch. Computer Vision with Deep Learning using Python. https://www.pyimagesearch.com/deep-learning-computer-vision-python-book, May 2019.

[RB01] C. A. Ruiz and M. S. Basualdo. Redes Neuronales: Conceptos Básicos y Aplicaciones. https://www.frro.utn.edu.ar/repositorio/catedras/quimica/5_anio/orientadora1/monograias/matich-redesneuronales.pdf, March 2001.

[RDGF16] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, Las Vegas, USA, June 2016.

[RF18] J. Redmon and A. Farhadi. YOLOv3: An Incremental Improvement. *arXiv e-prints*, April 2018.

[Sch97] R. R. Schaller. Moore's law: past, present and future. *IEEE Spectrum*, 34(6):52–59, June 1997.

[SMA+19] M. Simon, S. Milzy, K. Amende, A. Krasu, J. Honer, T. Sämann, H. Kaulbersch, S. Milz, and H. M. Gross. Complexer-YOLO: Real-Time 3D Object Detection and Tracking on Semantic Point Clouds. *arXiv e-prints*, April 2019.

[SMAG18] M. Simon, S. Milzy, K. Amendey, and H. Gross. Complex-YOLO: An Euler-Region-Proposal for Real-time 3D Object Detection on Point Clouds. In *The European Conference on Computer Vision (ECCV) Workshops*, Munich, Germany, September 2018.

[SVI+16] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception Architecture for Computer Vision. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, Las Vegas, USA, June 2016.

[TDHL18] X. Tang, Daniel K. Du, Z. He, and J. Liu. PyramidBox: A Context-assisted Single Shot Face Detector. In *The European Conference on Computer Vision (ECCV)*, pages 797–813, Munich, Germany, September 2018.

[Tur50] A. M. Turing. I.—computing machinery and intelligence. *Mind*, LIX(236):433–460, 10 1950.

[XZYA18] W. Xiang, D. Zhang, H. Yu, and V. Athitsos. Context-Aware Single-Shot Detector. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1784–1793, Nevada, USA, March 2018.

[YLLT16] S. Yang, P. Luo, C. C. Loy, and X. Tang. WIDER FACE: A Face Detection Benchmark. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5525–5533, Las Vegas, USA, June 2016.

[ZQX+18] Z. Zhang, S. Qiao, C. Xie, W. Shen, B. Wang, and A. L. Yuille. Single-Shot Object Detection With Enriched Semantics. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5813–5821, Utah, USA, June 2018.

[ZWB+18] S. Zhang, L. Wen, X. Bian, Z. Lei, and S. Z. Li. Single-Shot Refinement Neural Network for Object Detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4203–4212, Utah, USA, June 2018.

[ZWL97] P. Zhan, S. Wegmann, and S. Lowe. Dragon Systems' 1997 Mandarin Broadcast News System. In *in DARPA Broadcast News Workshop*, pages 9–17, Virginia, USA, March 1997.